Primitive Operation Aggregation Algorithms for Improving Taxonomies for Large-Scale Hierarchical Classifiers

Kiyoshi Nitta
Yahoo Japan Research
Midtown Tower, 9-7-1 Akasaka, Minato-ku, Tokyo, Japan
Email: knitta@yahoo-corp.jp

***Abstract*—Naive implementations of hierarchical classifiers that classify documents into large-scale taxonomy structures may face the contradiction between relevancy and efficiency performances. To address this problem, we focused on taxonomy modification algorithms for gradually improving the relevance performances of large-scale hierarchical classifiers. We developed four taxonomy modification algorithms that aggregate primitive operations before investigating hierarchical relevance performances. All but one produced taxonomy sequences that generate classifiers exhibiting practical efficiencies. One algorithm, which strictly maintains balanced proportions of taxonomy structures, generated a taxonomy sequence producing classifiers that exhibit stable relevancy performances. Another algorithm, which roughly maintains the proportions of taxonomy structure, but strictly maintains the maximum size of the training corpus for each local classifier, generated a taxonomy producing a classifier that exhibited the best relevance performance in our experiment. The base classification system we developed for this experiment uses an approach that locates local classifiers per parent node of taxonomies. It is able to classify documents into directed-acyclic-graph structured taxonomies. The system reached the level of practical hierarchical classification systems that efficiently and relevantly predict documents into over 10,000 taxonomy classes.**

***Keywords*-hierarchical classification; taxonomy modification**

I. INTRODUCTION

A classifier with a larger set of classes has the potential to enable more precise predictions than one with fewer classes. Hierarchical structures are commonly applied to large sets of classes to increase the usability of the classes. A hierarchical structure for a classifier is called a *taxonomy*. Most taxonomies are constructed manually. They provide natural and easy-to-use methods for users to access categorized documents. Automatic classification systems that classify documents into hierarchical taxonomies are called *hierarchical classifiers*. What strategy we should take to implement hierarchical classifiers is one of the most essential aspects when classifying documents into large-scale taxonomies. There are three basic classification strategies: the *top-down (or local) strategy* [1]–[6], the *big-bang (or global) strategy*, and the *flat strategy* [7].

While taxonomies usually provide natural and easy-to-use methods for users, they rarely provide the best efficiency and relevancy hierarchical structures for automatic classifiers. There are two approaches for improving hierarchical classifier performances by maintaining the benefit of manually constructed taxonomies:

1) *clustering approach* — decomposing a taxonomy into a flat set of classes and applying an ordinary clustering algorithm to those classes for building the best performance hierarchical structure [8],
2) *gradually modifying approach* — gradually modifying taxonomy structures by applying promote, demote, and merge primitive operations on hierarchical structures [9]–[11].

The first approach has a high possibility of obtaining the best performing hierarchical structure because there is no restriction in arranging structures. The second approach has an advantage that it can precisely control the balance of modified hierarchical structures. Unbalanced hierarchical structures may result in producing inefficient or irrelevant classifiers, especially when they used the local classification strategy.

We present four versions of taxonomy modification algorithms formulated based on the gradually modifying approach for local strategy hierarchical classifiers. The first algorithm simply and simultaneously accumulates primitive operations for all taxonomy nodes. The second algorithm reduces primitive operations that induce child node concentrations to specific nodes. The third algorithm limits the average depth of generating a taxonomy, while the fourth one also limits each corpus size used for training a parent node of the taxonomy. We developed an experimental classification system for evaluating these algorithms. We conducted an experiment with at most 120 iterations for each taxonomy modification algorithm. The result showed that the third algorithm produced stable relevance metric scores, while the fourth one produced much better relevance scores than the others.

Contributions of the paper can be summarized as follows:

- presentation of a taxonomy modification algorithm with stable relevancy performance,
- presentation of a taxonomy modification algorithm with best relevancy performance,

 ISBN: 978-1-61208-247-9

- implementation of practical hierarchical classifiers that take the "local classifier per parent" approach [7] to directed acyclic graph (DAG)-structured taxonomies, and
- implementation of practical hierarchical classifiers that efficiently and relevantly predict documents into over 10,000 taxonomy classes.

II. Large-Scale Hierarchical Document Classification

A. Hierarchical Classifier Specifications

General definitions and categorizations of machine-learning document classifiers have previously been published [12], as have the results of experiments on various types of hierarchical taxonomy classifiers [7], [13]–[15]. According to their applications and requirements, hierarchical classification systems might have different problems, algorithms, and corpus arrangements. This often causes confusion when the performance of various classifiers are compared. To guarantee fair comparisons with the classification system we developed for this experiment, we represent the specifications using the unifying framework for hierarchical classification [7], which provides comprehensive and essential notations of hierarchical classification tasks and solutions.

The specifications of our system are described as follows:

$$\begin{aligned} < \Gamma, \Psi, \Phi > &= < D, MPL, PD > \\ < \Delta, \Xi, \Omega, \Theta > &= < SPP, NMLNP, D, LCPN >, \end{aligned}$$

where $< \Gamma, \Psi, \Phi >$ and $< \Delta, \Xi, \Omega, \Theta >$ specify the types of problems and algorithms of the framework, respectively.

The property of original taxonomy data is mainly affected by the problem specifications. '$\Gamma = D$' indicates the DAG taxonomy graph structure. '$\Psi = MPL$' indicates that data instances can have multiple paths of labels, and '$\Phi = PD$' indicates that data instances may have labels corresponding to interim nodes of taxonomies (partial depth labeling). Further description of the data is explained in Subsection II-C.

Our primary goal was to develop practical web-document classifiers for large-scale web directories. This application category majorly affected the determination of the algorithm specifications. '$\Delta = SPP$' indicates that the algorithm performs single path predictions. Although multiple path predictions might help some types of users, they may complicate algorithms and lose classifier's efficiency. '$\Xi = NMLNP$' indicates that the algorithm performs non-mandatory leaf-node predictions. It is normal for web pages to be categorized not only in leaf nodes but also in interim nodes of web directory structures. '$\Omega = D$' indicates that the algorithm can handle DAG-structured taxonomies, and '$\Theta = LCPN$' indicates that the type of algorithm is 'local classifier per parent node'. The reasons for choosing this type is discussed in Subsection V-C.

B. System Implementation

We developed an experimental classification system that consists of eight modules: taxonomy translator, crawler, feature extractor, training and evaluation corpus generator, trainer, predictor, evaluator, and taxonomy modifier. The *taxonomy translator* translates the whole category structure of the original web directory data into a taxonomy that can be processed by the system. The *crawler* accesses the Internet, fetches the contents of all the documents (URLs) categorized in the taxonomy, and saves them as HTML files. The *feature extractor* first divides each crawled HTML file into four parts: the title, body, meta-keywords, and meta-description. It then selects a content string from each part, tokenizes the string into a set of terms, and saves them as bag of words (BOW) features. If the content is written in Japanese, the tokenization is performed by applying a Japanese morphological analyzer. The *training and evaluation corpus generator* reads relational information between the nodes and documents in the latest taxonomy and builds a relationship table that combines each taxonomy node with documents for training and evaluation purposes. The *trainer* trains a linear-kernel support vector machine (SVM) model of a multi-class classifier for each parent node of the latest taxonomy using the saved BOW features of training corpus documents. The *predictor* generates a BOW feature from a URL or loads a saved BOW feature of the URL, performs local classification, and predicts a taxonomy node class. The *evaluator* applies the predictor to the test corpus, saves all the classification results, and calculates several evaluation metrics. The *taxonomy modifier* generates a new taxonomy from the classification results saved by the evaluator. The next generation of the taxonomy can be produced by repeating the process sequence after the *training and evaluation corpus generator*.

The system is executed on a common Linux PC server that has four 2.33-GHz Xeon CPUs and 64-GB memory. Each CPU has 4 independent cores. The PC has totally 16 cores. The codes are written in Shell, Perl, and Erlang with a total of 25,532 lines.

C. Data Preparation

The taxonomy used in this experiment was constructed from the Yahoo! Japan category (http://dir.yahoo.co.jp/) snapshot on May 2, 2007. The taxonomy translator generated the original taxonomy, which had a total of 85,791 classes. The crawler fetched 490,018 documents representing 90.6% of all the documents in the taxonomy. The trainer produced 25,747 models for parent nodes of the taxonomy, each of which had more than one child node. The taxonomy modifier generated taxonomies that have more than 200,000 nodes in the experiment.

Our training example assigning policy can be formalized as follows using notations described by Fagni and Sebastiani

 ISBN: 978-1-61208-247-9

[16] and Cai and Hoffmann [7]:

$$Tr^{+}(c_j) = \bigcup_{c \in \Downarrow(c_j)} Tr^{+}(c) \setminus \bigcup_{c \in \Downarrow(\downarrow(\uparrow(c_j)) \setminus c_j)} Tr^{+}(c) \ . \quad (1)$$

Our classification, not taxonomy modification, algorithm used in the experimental hierarchical classifiers is classified as "local classifier per parent node" classification ($\Theta = LCPN$). Each parent classifier of this algorithm is a multi-class (not a binary) classifier. Only positive examples are explicitly assigned to class c_j for training its parent node $\uparrow (c_j)$ multi-class classifier. Because the handling taxonomy is a DAG structure ($\Omega = D$), child node class c_j may appear under different parent node classes ($c_{p1}, c_{p_2} \in \uparrow (c_j),\ c_{p1} \neq c_{p2}$). Sibling class sets $\downarrow (\uparrow (c_j)) \setminus c_j$ may differ under different parent nodes c_{p1} and c_{p2}. Therefore, the example set $Tr^{+}(c_j)$ under c_{p1} and $Tr^{+}(c_j)$ under c_{p2} are not always identical in the policy (1). This issue impacts many subsystems of hierarchical classifiers. Our classification system was designed to handle the issue by introducing example class identifiers, each of which includes not only the target class identifier but also the parent class identifier.

III. Improvement Methods for Large-Scale Taxonomy

A. Tang et al.'s Methods

The basic idea behind the methods proposed by Tang et al. [9] is that taxonomy improvements can be achieved by iterating three types of primitive operations (promote, demote, and merge) for taxonomy modifications. The promote operation raises a target node to the same hierarchical level as its parent node. The demote operation lowers a target node to the same hierarchical level as its child nodes as a sibling. The merge operation gathers two sibling target nodes into a newly created sibling node. The most conservative approach to taxonomy improvements might involve the following steps: 1) investigate hierarchical classifier relevance performances of taxonomies that can be produced by all possible single operations from the initial taxonomy, 2) select the operation and target nodes that perform the best, 3) apply the selected operation to the taxonomy, 4) take the modified taxonomy as the next initial taxonomy, and 5) repeat from step 1). Because this approach is far from efficient, Tang et al. proposed two heuristic methods [10] that perform more efficiently: a greedy approach, which selects possible single operations according to the classification result statistics of the initial taxonomy, and a local approach, which prioritizes target nodes according to the hierarchical structure of the initial taxonomy.

B. Accumulate Primitive Operations Algorithm

The two heuristic methods described above require training and evaluation processes for each taxonomy modified by an operation candidate before the operation is actually applied to the taxonomy. The total computational cost for those processes increases drastically according to the scale of the taxonomy. While the cost of training processes might be reduced by limiting the training node classifiers to only ones affected by single taxonomy modification operations, the cost of evaluation processes cannot be easily reduced. Local classification results might be affected by modifications of distantly located nodes, especially if those nodes belong to the upper positions in the taxonomy. The 'upper positions' mean the positions of taxonomy nodes located near the root node. Therefore, we developed four algorithms for modifying large-scale taxonomies by extending Tang et al.'s methods to drastically reduce the training and evaluation costs.

The first modification algorithm extended for large-scale taxonomies, *accumulate primitive operations (APO)*, consists of processes that find promote, demote, and merge operation candidates from all nodes in the initial taxonomy, solve conflicts among them, simultaneously apply those operation candidates to produce next-generation taxonomies, train a hierarchical classifier using the taxonomy, and evaluate the classifier. These processes reduce the number of the evaluation process iterations to one for generating the next taxonomy, while Tang et al.'s methods might iterate the evaluation process $O(n)$ times, where n is the size of the taxonomy.

C. Avoid Child Concentration Algorithm

Although the APO algorithm improved the efficiency of taxonomy improvement processes, it worsened training and predicting process efficiency. The later generations of taxonomies modified by the APO algorithm tend to concentrate child nodes to particular parent nodes. Nodes holding many classes and training documents require enough memory to maintain large number of support vectors in training and predicting processes of parent multi-class SVM classifiers. Such nodes also consume certain computational time. Therefore, we developed the *avoid child concentration (ACC)* algorithm to avoid the child concentration phenomena by extending the APO algorithm. The difference is that the new algorithm has an additional constraint in selecting promote candidates. The constraint limits each parent node to having at most a predefined number of child nodes.

D. Limit Average Depth Algorithm

The ACC algorithm solved the problem of training and predicting process efficiency by forcing the taxonomy structure not to spread widely. This results in deepening of the structure. Local classifiers have a weak point in that relevancy scores might worsen by accumulating errors through the path of parent node classifiers. The deeper structure enhances this weak point. Therefore, we developed the *limit average depth (LAD)* algorithm by restricting primitive operations that produce deeper structures. This algorithm

 ISBN: 978-1-61208-247-9

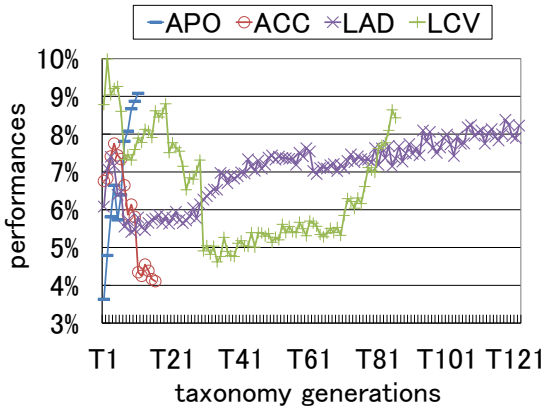

Figure 1. Flat f1 scores of classifiers generated using APO, ACC, LAD, and LCV algorithms

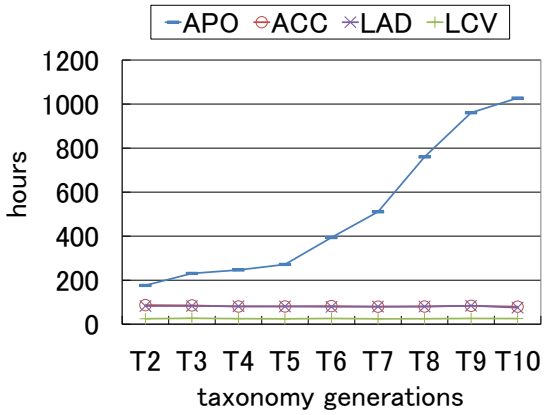

Figure 2. Predicting process efficiency of classifiers generated using APO, ACC, LAD, and LCV algorithms

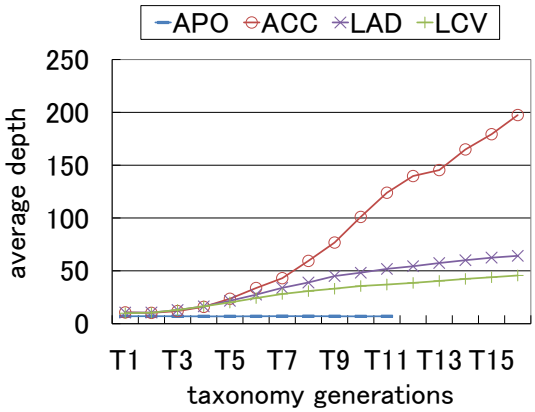

Figure 3. Average depths of taxonomies generated using APO, ACC, LAD, and LCV algorithms

differs from the ACC algorithm in definitions of the demote and merge candidate node set. The LAD algorithm may eliminate nodes, which are located at depths deeper than the predefined depth, from the candidates node set.

E. Limit Corpus Volume Algorithm

The LAD algorithm has two contradicting restrictions for selecting primitive operations. While the restriction for the number of child nodes deepens the taxonomy structure, the depth restriction attempts to increase the number of child nodes for several parent nodes. The contradicting restrictions decrease the number of primitive operation candidate nodes and require many taxonomy generations in order to improve relevance performances. Our detailed observation of taxonomy modification experiments revealed that the size of the training corpus affects training and predicting process efficiency more directly than the number of child nodes. Therefore, we developed the *limit corpus volume (LCV)* algorithm by primarily restricting the size of the corpus used for each parent node to train its classifier. Although it also applies the restrictions used in the LAD algorithm, the parameters are partly loosened for the primitive operation candidate sets to obtain more freedom to improve relevance efficiency.

IV. Empirical Results

Figure 1 shows flat f1 scores of hierarchical classifiers, whose companion taxonomies are generated by the APO, ACC, LAD, and LCV algorithms. On the X-axis, $T1$, $T2$, $T3$, $T4, \ldots$ denote generations for the 1st-, 2nd-, 3rd-, 4th-, ...taxonomies, respectively. The 'flat f1' scores are macro averaged f1 scores of taxonomy nodes, each of which has at least one expected document and one predicted document. The APO algorithm iteration was terminated at the 11th generation because the computational cost for training and predicting processes had became too large. The predicting process efficiencies for the first ten generations are shown in Figure 2. The ACC algorithm iteration was terminated at the 16th generation because the depth of the taxonomy structure had become too deep. The average depths of taxonomies generated by the APO, ACC, LAD, and LCV algorithms are shown in Figure 3. There is no significant reason for the termination of the LAD and LCV algorithms.

The metrics of flat precision, recall, and f1 scores might be too strict for relevance performances of hierarchical classifiers. More adequate metrics, hierarchical precision, recall, and f1 scores, have been proposed [7]. Because we had accidentally lost all raw classification results of the APO, ACC, and LAD algorithms, only the LCV algorithm results could be processed to calculate for those metrics. The hierarchical f1 scores of classifiers generated for the LCV algorithm are shown in Figure 4.

V. Discussion

A. Stability of Relevance Performance

The LAD algorithm generated a sequence of taxonomies, which produce hierarchical classifiers exhibiting stable relevance scores, as shown in Figure 1. Although each process

 ISBN: 978-1-61208-247-9

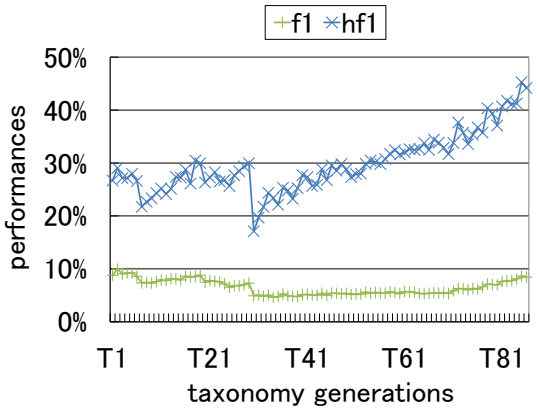

Figure 4. Flat f1 score (f1) and hierarchical f1 score (hf1) of classifiers generated using LCV algorithm

for generating a new taxonomy modifies a small number of nodes compared to other algorithms, the LAD flat f1 scores exhibited small improvements and reached the best record of such scores sometimes even after the hundredth generation. While the LAD algorithm has strict restrictions for solving the child node concentration and deep structure problems, it sometimes found a successful primitive operation candidate set. The LAD algorithm will perform well when that a) the original taxonomy structure is stable and b) the user has leeway to compute enough taxonomy modification iterations.

B. Best Relevance Performance

The LCV algorithm generated the 2nd generation taxonomy, which produced a hierarchical classifier that attained the best flat f1 score record (9.99%) of the experiment, as shown in Figure 1. The figure also indicates that the performance of this algorithm is not stable. There were several rapid depressions at the 7th, 12th, and 13th generations. The determinant reason for the depressions is currently under investigation. The major differences between the LCV from LAD algorithm were

1) avoiding corpus concentration and
2) loosening the parameter for child concentrations.

Differences 1) seems to play the same role as the restriction for avoiding child concentration, and succeeded in maintaining training and predicting process efficiency at the practical level, as shown in Figure 2. The effects of difference 2) seems to be both good, recording the best relevancy score, and bad, unsteadiness of relevancy scores of modified taxonomy sequences. The LCV algorithm should be applied when a) the original taxonomy structure is unstable and b) classification systems must be developed rapidly.

Figure 1 shows that the LCV algorithm performed the best at the 2nd generation measured using flat f1 scores. Figure 4 shows that the algorithm performed the best at the 85th generation measured using the hierarchical f1 scores. This means that the hierarchical classifier for the 85th generation taxonomy miss-classified more times than the 2nd taxonomy classifier, but the degree of miss-classification of the 85th one was less significant than the 2nd one. The flat f1 scores penalize miss-classifications uniformly. Nevertheless, the hierarchical f1 scores penalize them heavily if each one predicts a document to the more distanced node from its expected node. The distance-based loss scores [17] are commonly used to measure the degree of miss-classification. We could not use the loss scores successfully because it was difficult to combine the score with other relevancy scores such as f1. The hierarchical precision, recall, and f1 scores naturally combine relevance performance metrics with hierarchical penalty metrics. The criteria of selecting between the flat and hierarchical relevance metrics depends on the types of classification applications. If an application emphasizes the number of documents that were predicted to the exact matching nodes, the flat metrics should be used for evaluating classifiers. If another application tolerates miss-classification between nearby leaf-level sibling nodes, the hierarchical metrics should be used. We consider the web document classification applications as the latter type.

C. Classifier Design Selection

While we selected the 'local classifier per parent node' type classification algorithm ($\Theta = LCPN$), Silla and Freitas [7] presented other types as 'local classifier per node' (LCN), 'local classifier per level' (LCL), and 'global classifier' (GC). Implementations of the LCN algorithm locate local binary classifiers at all nodes that have the possibility of being predicted. The number of LCN local classifiers is always greater than that of the $LCPN$ algorithm. The local strategy hierarchical classifiers designed for large-scale taxonomies must have a large number of local classifiers. This is a fatal disadvantage for training and predicting process efficiency. Considering that we are using SVM-based multi-class local classifiers that consist of optimized combinations of binary classifiers, it might be feasible to apply the LCN algorithm with careful selection of the training corpus. The LCN classifiers tend to be trained using unbalanced corpus that consist of a few positive examples and a huge number of negative examples. It requires additional processing time to improve relevancy performance. The LCL and GC classifiers have to simultaneously treat many prediction classes. When they are applied for large-scale taxonomies, they increase the size of model, training time, and prediction latency.

D. Performance Limits of Large-Scale Hierarchical Classifiers

There are several issues to improving relevance and efficiency performances that have not been applied to our experimental hierarchical classifiers. While the classifiers might not exhibit the best relevance or efficiency, they seem to balance better relevance and efficiency performances

 ISBN: 978-1-61208-247-9

considering they deal with large-scale taxonomies and are executed on a commonly available server machine. The classifier design choices of the feature extraction (full BOW) and the classification algorithm ($\Theta = LCPN$) support balanced performance improvements. A large number of the improvements was achieved using the taxonomy modification method. The 11th generation taxonomy of the APO algorithm produced a classifier that resulted in 9.08% flat f1 value, as shown in Figure 1. Nevertheless, the efficiency of the classifier is inferior to all other evaluated hierarchical classifiers, as shown in Figure 2. One of the LCV classifiers exceeded the relevancy performance by maintaining practical efficiencies, as shown in Figure 1. The LCV classifier gives an example of practically executable large-scale hierarchical classifiers, each of whose taxonomy has more than 10,000 classes.

VI. Conclusion

We focused on taxonomy modification algorithms that gradually improve the relevance performances of large-scale hierarchical classifiers of web documents. Considering the research results from Tang et al. [9], [10], who took the same approach, we investigated and implemented four taxonomy modification algorithms that aggregate primitive operations before evaluating hierarchical relevance performances. The LAD algorithm generated a sequence of taxonomies that produce classifiers exhibiting stable relevancy performances. The LCV algorithm generated a taxonomy that produces a classifier that resulted in the best flat f1 score (9.99%) in our experiment. The hierarchical classifiers executed in the experiment used the "local classifier per parent" approach [7] and classified documents into DAG-structured taxonomies. The system we developed for this experiment reached the level of practical hierarchical classification systems that relevantly and efficiently predict documents into taxonomies containing more than 10,000 classes.

References

[1] M. Ceci and D. Malerba, "Classifying web documents in a hierarchy of categories: a comprehensive study," *J. Intell. Inf. Syst.*, vol. 28, no. 1, pp. 37–78, 2007.

[2] S. C. Gates, W. Teiken, and K.-S. F. Cheng, "Taxonomies by the numbers: building high-performance taxonomies," in *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*. New York, NY, USA: ACM Press, 2005, pp. 568–577.

[3] Y. Labrou and T. Finin, "Yahoo! as an ontology: using yahoo! categories to describe documents," in *CIKM '99: Proceedings of the eighth international conference on Information and knowledge management*. New York, NY, USA: ACM, 1999, pp. 180–187.

[4] D. Mladenic, "Turning yahoo into an automatic web-page classifier," in *Proceedings of the 13th European Conference on Aritficial Intelligence ECAI'98*, 1998, pp. 473–474.

[5] A. Sun and E.-P. Lim, "Hierarchical text classification and evaluation," in *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 521–528.

[6] D. Koller and M. Sahami, "Hierarchically classifying documents using very few words," in *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 170–178.

[7] C. N. Silla, Jr. and A. A. Freitas, "A survey of hierarchical classification across different application domains," *Data Min. Knowl. Discov.*, vol. 22, no. 1-2, pp. 31–72, Jan. 2011.

[8] T. Li, S. Zhu, and M. Ogihara, "Hierarchical document classification using automatically generated hierarchy," *J. Intell. Inf. Syst.*, vol. 29, no. 2, pp. 211–230, 2007.

[9] L. Tang, J. Zhang, and H. Liu, "Acclimatizing taxonomic semantics for hierarchical content classification from semantics to data-driven taxonomy," in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM Press, 2006, pp. 384–393.

[10] L. Tang, H. Liu, J. Zhang, N. Agarwal, and J. J. Salerno, "Topic taxonomy adaptation for group profiling," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 4, pp. 1–28, 2008.

[11] K. Nitta, "Improving taxonomies for large-scale hierarchical classifiers of web documents," in *Proceedings of the 19th ACM international conference on Information and knowledge management*, ser. CIKM '10. New York, NY, USA: ACM, 2010, pp. 1649–1652.

[12] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, 2002.

[13] S. Dumais and H. Chen, "Hierarchical classification of web content," in *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2000, pp. 256–263.

[14] T.-Y. Liu, Y. Yang, H. Wan, H.-J. Zeng, Z. Chen, and W.-Y. Ma, "Support vector machines classification with a very large-scale taxonomy," *SIGKDD Explor. Newsl.*, vol. 7, no. 1, pp. 36–43, 2005.

[15] A. Kosmopoulos, E. Gaussier, G. Paliouras, and S. Aseervatham, "The ecir 2010 large scale hierarchical classification workshop," *SIGIR Forum*, vol. 44, no. 1, pp. 23–32, Aug. 2010.

[16] T. Fagni and F. Sebastiani, "On the selection of negative examples for hierarchical text categorization," in *Proceedings of the 3rd Language Technology Conference (LTC 2007)*, 2007, pp. 24–28.

[17] L. Cai and T. Hofmann, "Hierarchical document categorization with support vector machines," in *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*. New York, NY, USA: ACM Press, 2004, pp. 78–87.

 ISBN: 978-1-61208-247-9