Hierarchy Evolution for Improved Classification

Xiaoguang Qi Brian D. Davison
Department of Computer Science & Engineering, Lehigh University
Bethlehem, PA 18015 USA
{xiq204,davison}@cse.lehigh.edu

ABSTRACT

Hierarchical classification has been shown to have superior performance than flat classification. It is typically performed on hierarchies created by and for humans rather than for classification performance. As a result, classification based on such hierarchies often yields suboptimal results. In this paper, we propose a novel genetic algorithm-based method on hierarchy adaptation for improved classification. Our approach customizes the typical GA to optimize classification hierarchies. In several text classification tasks, our approach produced hierarchies that significantly improved upon the accuracy of the original hierarchy as well as hierarchies generated by state-of-the-art methods.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*

General Terms

Algorithms, Performance

Keywords

Topical categorization, Hierarchical classification, SVM

1. INTRODUCTION

Classification can be performed based on a flat set of categories, or on categories organized as a hierarchy. In flat classification, a single classifier learns to classify instances into one of the target categories. In hierarchical classification, a separate classifier is trained for each non-leaf node in the hierarchy. During training, each classifier is trained to categorize the instances that belong to any of the descendants of the current node into its direct subcategories. When deployed, an instance is first classified by the root classifier, and then passed to one of the first level categories with the highest probability. This process is repeated iteratively from top to bottom, invoking one classifier at each level, until reaching a leaf node. Prior work has shown that hierarchical classification has performance superior to that of flat classification (e.g., [3, 5, 1]).

Hierarchical classification is typically performed utilizing human-defined hierarchies, reflecting a human view of the domain. However, these hierarchies are usually created without consideration for automated classification. As a result, hierarchical classification based on such hierarchies is unlikely to yield optimal performance.

In this paper, we propose a new method based on genetic algorithms to create hierarchies better suited for automatic classification. In our approach, each hierarchy is considered to be an individual. Starting from a group of randomly generated seed hierarchies, genetic operators are randomly applied to each hierarchy to slightly reorganize the categories. The newly generated hierarchies are evaluated and a subset that are better fitted for classification are kept, eliminating hierarchies with poor classification performance from the population. This process is repeated until no significant progress can be made. In our experiments on several text classification tasks, our algorithm significantly improved the classification accuracy compared to the original hierarchy and also outperformed state-of-the-art adaptation approaches. Our contributions include:

- a new approach to improved classification by hierarchy adaption; and,
- adaptation of genetic operators on hierarchies and an analysis of their utility.

The basic idea of generating/adapting hierarchies for better classification is not new. Li et al. [4] proposed a method of using linear discriminant projection to generate hierarchies. In this approach, all the documents within the hierarchy are first projected onto a lower dimensional space. Then, the leaf categories are clustered using hierarchical agglomerative clustering to generate the hierarchy. Tang et al. [8] proposed a method for hierarchy adaptation by iteratively checking each node in the hierarchy, and making slight modifications locally. This approach can also be used to model dynamic change of taxonomies [7]. Additional related work may be found in our full report [6].

There are at least two main differences that separate our approach from previous work:

- After each iteration, we keep a comparatively large number of best performing hierarchies rather than only keep the best one and discard the rest; we will show later that the best hierarchy does not always come from an adaptation of the previous best hierarchy.
- Unlike previous approaches that gradually adapt a hierarchy by making a slight change at a step, the crossover operators we customize for hierarchy evolution allow a new hierarchy

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

to inherit characteristics from both parents, so that it is significantly different from either parent. This will take previous approaches many more iterations to achieve, or perhaps unachievable at all because of greedy search strategies.

2. APPROACH

As introduced previously, hierarchical classification can often perform better than flat classification. However, this is not always true.It depends on the choice of hierarchy. In order to further motivate this work using real-world data, we randomly selected 7 leaf categories containing 227 documents from LSHTC dataset (see Section 3.1 for details), exhaustively generated all the possible hierarchies based on the selected categories, and tested the classification performance for every hierarchy. In total, 39,208 hierarchies were generated, out of which 48.2% perform worse than flat classification in terms of accuracy. This verifies our intuition that improving hierarchical classification needs a well-chosen hierarchy. In the following, we will describe how to adapt genetic algorithms to search for a better hierarchy.

2.1 Overview

A genetic algorithm (GA) is a search/optimization method that resembles the evolution process of organisms in nature. Like any other search method, it searches through the solution space of a problem, looking for an optimal solution. In our problem, we consider each possible hierarchy to be a solution (or an individual in GA, specifically). As illustrated by the small example above, our solution space is often too large to perform an exhaustive search except for very small datasets.

A typical GA usually starts with an initial population of individuals, then iteratively repeats the following search procedure until the stopping criterion is satisfied. In each iteration, a subset of individuals is selected for reproduction by applying mutation and crossover operations on them. Then the fitness of each new individual is evaluated, and low-fitness individuals are dropped, leaving a better fitted population as the start of next iteration. In our approach, we will leave the high level procedure of GAs as described above unchanged, while adapting representation method and reproduction operators to make them fit the hierarchical classification problem. We chose a GA instead of other generic search methods for at least two reasons. First, it intrinsically supports large populations rather than greedy, single path optimization. Second, we can adapt its reproduction operators to allow significant changes to the solutions without changing the high level search procedure.

2.2 Hierarchy representation

In our work, each hierarchy is represented as a sequence of numeric IDs and parentheses, in which each ID corresponds to a leaf category, and each pair of parentheses represents a more generic category consisting of the categories in between them. Multiple levels in the hierarchy are reflected using nested parentheses.

More formally, we represent a hierarchy using the following rules:

1. Each leaf node is represented by its numeric id;
2. Each non-leaf node is represented by a list of all its children nodes enclosed in a pair of parentheses;
3. The hierarchy is represented recursively using Rule 1 and 2.
4. The outermost pair of parentheses is omitted.

Figure 1 illustrates a small example, which will be represented as (1 2 5) ((3 6) 4).

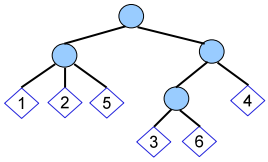

Figure 1: A small hierarchy example.

The hierarchy representation method above serializes a hierarchical tree into a sequence of tokens. However, different representations may correspond to the same hierarchy. For example, (1 2 5) ((3 6) 4) and (2 1 5) ((6 3) 4) define the same hierarchy. We detect duplicate hierarchies using a two-step canonicalization. We first remove unnecessary parentheses, then sort siblings according to the minimal class id in its subtree (refer to our report [6] for details).

2.3 Reproduction

2.3.1 Mutation

In typical GA, mutation is performed by switching a random bit in the chromosome string. However, this operation is not as straightforward in the hierarchical setting. We design three mutation methods that are suitable for hierarchy evolution: promotion, grouping, and switching.

The *promotion* operator randomly selects a node n (which can be either a leaf or non-leaf node), and promote n as a child of its grandparent, i.e., n becomes a sibling of its parent node. The *grouping* operator randomly selects two sibling nodes and groups them together. If a non-leaf node n has k children, $C = \{c_i | i = 1..k\}$, where $k \geq 2$. We randomly select two nodes c_x and c_y from c_1 through c_k, and remove them from C. Then we add a new node c_z into C so that c_z becomes a child node of n. Finally, we make c_x and c_y children of c_z. The *switching* operator randomly selects two nodes m and n (and the subtrees rooted at those locations) in the whole hierarchy, and switches their positions.

2.3.2 Crossover

In a typical GA, crossover is performed by swapping segments of the chromosome string between the parents. In the hierarchical setting, however, directly swapping parts of the hierarchy representation will generate invalid hierarchies. Therefore, we need crossover methods customized for hierarchy evolution.

We used two types of methods: swap crossover and structural crossover. The two parents are noted as h_{p1} and h_{p2}, the children h_{c1} and h_{c2}. In *swap crossover*, a child h_{c1} is generated using the following steps. First a split point p is randomly chosen in h_{p1}. We note the the part starting from the beginning of the representation string to the split point p as h'_{p1}. We remove the segment after p from h_{p1} and only consider h'_{p1}. Then right parentheses are added at the end to balance with the existing left parentheses. Suppose S is the set of leaf nodes that appear in h'_{p1}; we go through h_{p2} and remove all the nodes n if $n \in S$. This removal transforms h_{p2} into h'_{p2}. Finally, h'_{p1} and h'_{p2} are concatenated to form h_{c1}. The other child h_{c2} is generated by switching h_{p1} and h_{p2} before applying the above procedure. Figure 2 illustrates the process of swap crossover.

For our second approach, we note that a hierarchy can be seen as an integration of two independent characteristics: the tree structure and the placement of leaf nodes. At a high level, *structural crossover* aims to "mix and match" these two factors. A child hierarchy inherits the structural information from one parent, and

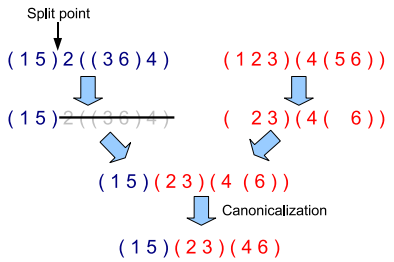

Figure 2: An example of swap crossover.

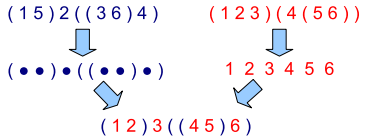

Figure 3: An example of structural crossover.

placement of leaf nodes from the other. In our implementation, h_{c1} is generated using the following method. First, every leaf node in h_{p1}'s representation is replaced with a blank space. Then these blank spaces are filled with the leaf nodes in h_{p2} using the order that they appear in h_{p2}. h_{c2} is generated by switching h_{p1} and h_{p2}. Figure 3 illustrates the process of structural crossover.

A GA requires a fitness function for evaluating new individuals. We define the fitness of a hierarchy as the classification accuracy using this hierarchy evaluated on validation data. A GA also needs a stopping criterion to terminate the iterative evolution process. In our algorithm, we keep a watch list of the top N_{watch} best hierarchies. If the performance of the top hierarchies does not change between two consecutive iterations, the algorithm terminates. In the following experiments, we set N_{watch} to 5.

3. EXPERIMENTS

3.1 Experimental setup

We used three public datasets to test our algorithm. The first two datasets are from the first Large Scale Hierarchical Text Classification (LSHTC) challenge[1] held in 2009. We selected a toy dataset from Task 1 with 36 leaf categories and 333 documents, which will be referred to as LSHTC-a. We also used the dry-run dataset from Task 1, which has 1,139 leaf categories and 8,181 documents. We will refer to this dataset as LSHTC-b. Both datasets are partitioned into three subsets: a training set used to train classifiers during the training process, a validation set used to estimate the fitness score of each generated hierarchy, and a test set to evaluate the final output hierarchy. The third dataset is WebKB[2], containing 7 leaf categories and 8,282 documents. We performed four-fold cross-validation on WebKB. LibSVM [2] is used as the base classifier to implement the standard hierarchical SVM. We used all the default settings in LibSVM, including the radial basis function kernel as it yields better performance than the linear kernel on the datasets we used. In our algorithm, we set the population size to 100 on LSHTC-a and WebKB, 500 on LSHTC-b.

We compared our approach (Evolution) with two existing state-of-the-art approaches: a hierarchy adaptation approach called Hierarchy Adjusting Algorithm (HAA) [8], and a hierarchy generation approach called Linear Projection (LP) [4]. We implemented HAA according to the algorithm outlined in Figure 12 and 13 of [8]. All three search methods were implemented. In our implementation of HAA, we set the stopping criterion to 0.001. That is, when the improvement between two consecutive iterations is less than 0.001, the algorithm terminates. We also compared our algorithm with Linear Projection on WebKB dataset. Instead of re-implementing the LP algorithm, we directly used the automatically generated hierarchy on WebKB reported in the Linear Projection paper, and performed the four-fold cross-validation based on that hierarchy.

[1]http://lshtc.iit.demokritos.gr/node/1
[2]http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/

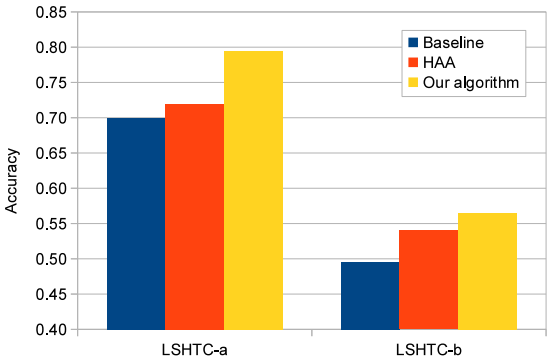

Figure 4: Accuracy on LSHTC datasets compared across different methods.

3.2 Experimental results

On LSHTC-a, our algorithm converged after seven iterations, improving the performance on the test set from 70% to 79.5% (averaged across three runs using different random seeds). HAA converged after two iterations with the accuracy improved to 71.9%. On LSHTC-b, it took 49 iterations for our algorithm to converge, and 18 iterations for HAA. HAA improved upon the baseline's accuracy of 49.6% to 54.0% , while our algorithm's final output hierarchies have an average accuracy of 56.5% across three runs using different random seeds. The results are shown in Figure 4.

On WebKB, we compared our algorithm with flat classification, HAA, and Linear Projection. The flat classification has an accuracy of 70.3% averaged across the four folds. Linear Projection and HAA improve the accuracy to 74.6% and 75.4%, respectively. Our algorithm further improves to 76.4%, a 21% reduction in error rate compared with flat classification. Figure 5 shows the average performance and standard deviation for each method. Two-tailed t-tests showed the improvement of our algorithm over other algorithms is statistically significant.

3.3 Analysis

As we pointed out previously, our approach differs from existing hierarchy adaptation approaches from at least two aspects: making significant changes to hierarchies and a larger population size to maintain population variety. Now we analyze quantitatively whether these differences make our approach outperform existing methods. The following analysis is performed on LSHTC-b.

In order to evaluate the effectiveness of the genetic operators, we calculate the improvement that each type of operator brings to a hierarchy. Figure 6 shows the average improvement in terms of accuracy. On average, all the five operators have a negative impact on the accuracy. For example, the "swap crossover" even decreases accuracy by 0.018 when averaged across all evolution

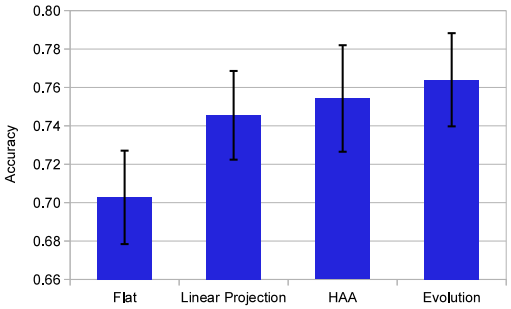

Figure 5: Accuracy on WebKB dataset compared across different methods.

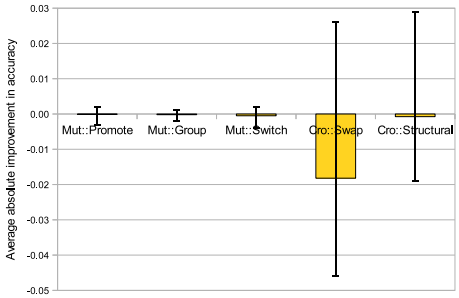

Figure 6: Improvement compared across different genetic operators.

operations. That is, on average, every time a "swap crossover" is applied on a hierarchy, the newly generated hierarchy has an accuracy lowered by 0.018. Fortunately, the genetic algorithm only keeps the best hierarchies in the population, and discards the rest. Therefore, the degradation is counter-balanced by such a selection process, making an overall increasing trend (as we showed previously in experimental results). We also calculated the standard deviation of the improvement, as well as minimal and maximal improvement. The error bars in Figure 6 show the minimal and maximal improvement. The standard deviation of the operators are 0.0004, 0.0005, 0.0009, 0.0100, 0.0028, respectively. This indicates that the mutation operators only have slight impact on the accuracy while changes made by crossover are more significant. In the best case, "swap crossover" improved accuracy by 0.026, "structural crossover" improved 0.029, while all the mutation operators can only improve no more than 0.002 at their best. These statistics match our intuition that more significant changes can potentially bring better improvements than local modifications.

Unlike previous approaches that only carry the best hierarchy into the next iteration, we keep hundreds of hierarchies in the population. This raises a reasonable concern about our approach: is the large population necessary? To answer this question, we first identified the top-N hierarchies at each iteration, then back-traced their parents from the previous iteration, and finally found out the ranks for their parents. The results are plotted in Figure 7. The x-axis shows the range of the parents' ranking. For example, range 1 corresponds to rank 1 through 50. Range 10 corresponds to rank 451 through 500. We can see that most parents rank above 300, while still leaving a significant portion between 300 and 500. From another point of view, if we were to keep only 50 hierarchies, we would have lost approximately 88% of the top 1 hierarchies at each iteration, 84% of the top 5 and top 10. Although we could probably

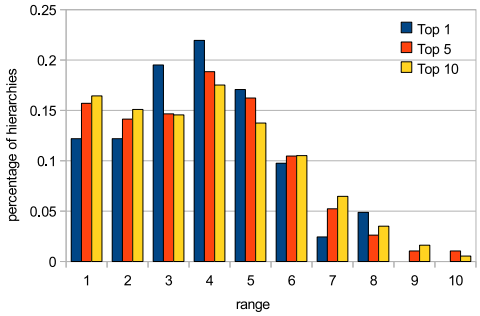

Figure 7: Parent rankings of top hierarchies.

shrink the population size by 20% without significant degradation in accuracy, this supports the idea that the comparatively large population is necessary.

4. CONCLUSION

In this paper, we proposed a hierarchy adaptation approach using the standard mechanisms of a genetic algorithm along with special genetic operators customized for hierarchies. Unlike previous approaches which only keep the best hierarchy in the search process and modify the hierarchy locally at each step, our approach maintains population variety by allowing simultaneous evolution of a much larger population, and enables significant changes during evolution. Experiments on multiple classification tasks showed that the proposed algorithm can significantly improve automatic classification, outperforming existing state-of-the-art approaches. Our analysis showed that the variety in population and customized reproduction operators are important to improvement in classification.

Acknowledgments

We thank Dr. Hector Munoz-Avila for discussions that inspired the use of genetic algorithms, and Liangjie Hong and Na Dai for helpful discussions. This material is based upon work supported by the National Science Foundation under Grant Number IIS-0545875.

5. REFERENCES

[1] P. N. Bennett and N. Nguyen. Refined experts: Improving classification in large taxonomies. In *Proc. 32nd Annual Int'l ACM SIGIR Conf. on Research and Dev. in Info. Retr.*, pages 11–18, 2009.

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

[3] S. Dumais and H. Chen. Hierarchical classification of web content. In *Proc. 23rd Annual Int'l ACM SIGIR Conf. on Research and Dev. in Information Retr.*, pages 256–263, 2000.

[4] T. Li, S. Zhu, and M. Ogihara. Hierarchical document classification using automatically generated hierarchy. *J. Intell. Inf. Syst.*, 29:211–230, October 2007.

[5] 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 Explorations Newsletter*, 7(1):36–43, 2005.

[6] X. Qi and B. D. Davison. Optimizing hierarchical classification through tree evolution. Technical Report LU-CSE-11-002, Dept. of Computer Science and Engineering, Lehigh University, 2011.

[7] L. Tang, H. Liu, J. Zhang, N. Agarwal, and J. J. Salerno. Topic taxonomy adaptation for group profiling. *ACM Trans. Knowl. Discov. Data*, 1:1:1–1:28, February 2008.

[8] L. Tang, J. Zhang, and H. Liu. Acclimatizing taxonomic semantics for hierarchical content classification. In *Proc. of the 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 384–393, 2006.