Hierarchical Classification of Documents with Error Control

Chun-hung Cheng[1], Jian Tang[2], Ada Wai-chee Fu[1], and Irwin King[1]

[1] Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, Hong Kong
{chcheng,adafu,king}@cse.cuhk.edu.hk
[2] Department of Computer Science
Memorial University of Newfoundland, St. John's, NF, A1B 3X5 Canada
jian@cs.mun.ca

Abstract. Classification is a function that matches a new object with one of the predefined classes. Document classification is characterized by the large number of attributes involved in the objects (documents). The traditional method of building a single classifier to do all the classification work would incur a high overhead. Hierarchical classification is a more efficient method — instead of a single classifier, we use a set of classifiers distributed over a *class taxonomy*, one for each internal node. However, once a misclassification occurs at a high level class, it may result in a class that is far apart from the correct one. An existing approach to coping with this problem requires terms also to be arranged hierarchically. In this paper, instead of overhauling the classifier itself, we propose mechanisms to detect misclassification and take appropriate actions. We then discuss an alternative that masks the misclassification based on a well known software fault tolerance technique. Our experiments show our algorithms represent a good trade-off between speed and accuracy in most applications.

Keywords: Hierarchical document classification, naive Bayesian classifier, error control, class taxonomy, parallel algorithm

1 Introduction

Classification is a function that matches a new object with one of the predefined classes. A special kind of classification, *document classification*, has recently caught researchers' attention [4,12,20]. A document classifier categorizes the documents into the classes based on their content. This problem is characterized by the large number of attributes involved in the objects (documents). While a few hundred attributes are considered as very big for a traditional classifier, documents often contain thousands or even tens of thousands of terms. The traditional method of building a single classifier for all the classification work, known as *flat classification*, would incur a high overhead.

Koller and Sahami [12] propose the use of hierarchical classification in this context. Instead of a single classifier, a set of classifiers distributed over a *class* *taxonomy* are used, one for each node. A document is classified in a top-down fashion from the root to the leaf. For each current node (i.e. class), the child of maximum likelihood is selected. Thus, by decomposing a job into smaller jobs like this and some other techniques (e.g. feature selection), the amount of work can be maintained at a manageable level. This method is called *simple hierarchical classification* in this paper. However, once a misclassification occurs at a high level node, there is little chance to accommodate it at the low levels. The deeper the classification goes, the further it drifts away from the correct one. A variation of simple hierarchical classification, known as *TAPER*, is proposed in [4]. To avoid misclassification, it attempts to search for a global optimal probability by assigning the probability to the edge of the taxonomy graph in some ways that would transform the search into a least-cost path problem.

Weiss and Kulikowski [20] propose a different scheme of which one of the main goals is to remedy the misclassification problem. They utilize a single classifier over 'global' terms. The classifier is actually a set of special kind of association rules whose right sides are class labels, but only a portion of the rules are selected for the classification. A problem with this scheme is that in addition to class hierarchy, a term hierarchy is also required, which does not always exist. Also it is not clear if the selection rule can adequately reduce the number of association rules to make the job by the lone classifier manageable in the general case.

In this paper, we attack this misclassification problem from a different angle. We adopt hierarchical classification model due to its efficiency, but instead of trying to reduce the misclassification rate by overhauling the classifier itself, we develop mechanisms to detect the misclassification as early as possible and then take appropriate actions. We also discuss an alternative that masks the misclassification using a well known software fault tolerance technique.

The rest of this paper is organized as follows. In Section 2, we present a general model for document classification using hierarchical classifiers. In Section 3, the two error control schemes are introduced. We move to the experimental results in Section 4 and finally, we conclude this paper in Section 5.

2 Document Classification

Informally, a document is a pattern which consists of a number of terms and is attached with a class value (topic). Each term can occur multiple times in a document. The dependencies between the class values and the terms follow certain probabilistic distribution.

More specifically, we adopt a naive Bayesian model from [4]. Each class c is associated with a multinomial term-variable V_c. V_c can take values i, $1 \leq i \leq n_c$, with probability $p_{i,c}$ where each i denotes a term and n_c is the total number of different terms. A document in a class is then modeled as a collection of values (duplicates allowed) that the associated variable V_c generates successively. Let d be a given document in class c, h_d be its length, $z_{i,d}$ be the number of occurrences of value i in d, and $z_c = \sum_{d \in c, j=1,2,\cdots,n_c} z_{j,d}$. Let $P(d \mid c)$ be the probability that a randomly chosen document is d given that it is in c. Then we have

$$P(d \mid c) = \frac{h_d!}{z_{1,d}! z_{2,d}! \cdots z_{n_c,d}!} \Pi_{j=1}^{n_c} p_{j,c}^{z_{j,d}}. \tag{1}$$

where the value of $p_{i,c}$ can be estimated as $\frac{(\sum_{d \in c} z_{i,d})+1}{z_c+n_c}$. It is not the more intuitive value $\frac{\sum_{d \in c} z_{i,d}}{z_c}$. See [18] for a justification.

Let T be the class taxonomy, c be an internal node and c_i where $1 \leq i \leq q$ be the ith child of c. Given a document d, the classifier at node c classifies it into one of $c_1, \cdots, c_q$ by choosing c_i that maximizes $P(c_i \mid c, d)$, the probability of d belonging to c given it belongs to c'.

$$P(c_i \mid c, d) = \frac{P(c_i, d)}{P(c, d)} = \frac{P(c_i) P(d \mid c_i)}{\Sigma_{j=1}^{q} P(c_j) P(d \mid c_j)} \tag{2}$$

where $P(c_i, d)$ is the probability that we are given a document d and d belongs to c_i; $P(d \mid c_j)$ is estimated as stated above and $P(c_k)$ can be estimated as the fraction of the number of the documents that belongs to class c_k.

Since documents can contain a large number of terms, we must perform *feature selection* to reduce the cost. In addition, it can separate unindicative terms, or *noise*, from feature terms and increase accuracy of the classifier, since too many features may cause overfitting and loss of generality. As described in [12], features are context sensitive, meaning that we have different features at different splits in the taxonomy. Thus feature selection should be carried out at each split in the taxonomy.

One way to do feature selection is to use *Fisher Index* [4]. Let t_k be the kth term, $w(t_k, d)$ be the relative frequency of term t_k in document d, and $aw(t_k, c)$ $= \frac{1}{|c|} \Sigma_{d \in c} w(t_k, d)$. Thus, the Fisher Index of t_k for class c is:

$$Fisher(t_k, c) = \frac{\Sigma_{i=1}^{\ell} \mid c_i \mid (aw(t_k, c_i) - aw(t_k, c))^2}{\Sigma_{i=1}^{\ell} (\frac{1}{|c_i|} \Sigma_{d \in c_i} (w(t_k, d) - aw(t_k, c_i))^2)} \tag{3}$$

The idea is that a smaller value for the denominator implies a closer distance along dimension t_k among the points within each class, and a larger value for the numerator signifies a larger distance between any class and c. Thus a larger Fisher Index indicates a larger discriminative power of a term for a class. Let L be the list of terms in the descending order of their Fisher Indexes for c. We pick up a prefix F of L, and use F for the classification for c. Since F leaves out most noise terms, it reduces misclassification. The number of terms in F, known as the *feature length*, is a choice by users.

3 Error Control Schemes

Since simple hierarchical classification is problematic when a misclassification occurs at an early level, our approach is to incorporate error control mechanisms into the algorithm. We propose two schemes, namely recovery oriented error handling and error masking. The latter is a parallel algorithm and should run on a multi-processor machine.

3.1 Recovery Oriented Error Handling

The recovery oriented error handling approach is inspired by the way a transactional database is recovered upon failure. When a failure occurs in a transactional database, a previous consistent state is reconstructed, and an appropriate recovery action is taken based on that state. To bring the idea of database recovery to document classification, a consistent state here means an ancestor class node to which a document is classified with high confidence. We call it a *High Confidence Ancestor* (HCA). When a document is misclassified into a wrong path in the class taxonomy, we can restart from the HCA and then select another path. However, from our empirical studies rollback and reclassification are very time consuming. To simulate the effects of recovery, we try to identify the *wrong paths* first and avoid them during the classification.

To detect the wrong paths, we associate each document with a value called *closeness indicator* (CI) to indicate how close the document is to a given topic. Once a document is misclassified, the more it descends along the selected path, the further it would drift away from the distribution represented by the nodes in the path. When CI drops below a certain threshold, we may conclude that we are on the wrong path. For example, consider the class taxonomy depicted in Fig. 1. Assume a document is about ‘folk dance’, but has been misclassified into ‘Business’. While this may seem not entirely unacceptable, it would be less acceptable to classify it into either ‘Financial’ or ‘Insurance’. Suppose it is classified into ‘Financial’ by the classifier at ‘Business’ node. Then it faces the choices of ‘Investment in stock market’ and ‘Portfolio arrangement of mutual funds’. Neither of these is remotely related to ‘folk dance’, so CI would fall to a small value and the path would be rejected.

Clearly, CI should be calculated without referring to the probabilities we used in the classification. Therefore, instead of the one-step probability, the probability of d belongs to c given the HCA is used as CI. Let c be the class that document d has been classified into. Let c' be the HCA of c for d. The CI of d with respect to c under c' is computed as:

$$CI(d, c \mid c') = P(c \mid d, c') = \frac{P(c, d \mid c')}{P(d \mid c')} = \frac{P(c \mid c')P(d \mid c)}{\Sigma_{i \in c'} P(i \mid c')P(d \mid i)}. \tag{4}$$

A simple way to determine the threshold is to use $1/N$, where N is the total number of classes at the same level as c and leaf classes at some level above c[1], in the subtree rooted at c'.

We maintain a moving window of l levels where l is a user parameter. The top and the bottom of the window correspond respectively to the levels of the current HCA and the class into which the document is being classified. Initially, the HCA is the root. The window moves downwards one level when the class at the bottom edge passes the test by CI, resulting in a new HCA at one level lower than it was prior to the move of the window.

[1] The class taxonomy can be an unbalanced tree.

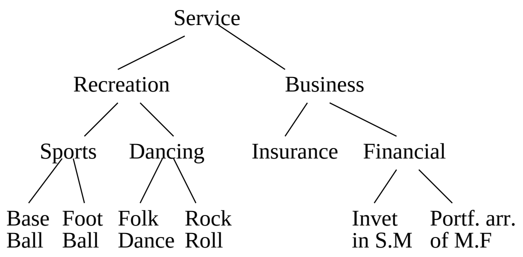

Fig. 1. A class taxonomy

```
Algorithm hc_recovery_oriented($T, d, l$)
// $T$: class taxonomy
// $d$: document to be classified
// $l$: difference of the level of HCA and that of the current node
1.  $HCA \leftarrow$ root($T$)
2.  Loop
3.      $CI\_list \leftarrow$ find_CI_list($HCA$, $l$)
4.      If no_of_element($CI\_list$) = 1 then
5.          $result\_class \leftarrow$ only element of $CI\_list$
6.      Else
7.          $result\_class \leftarrow \arg\max_{c \in CI\_list}$ {local_prob($HCA, c$) }
8.      Endif
9.      If $result\_class$ is leaf Then return $result\_class$
10.     $HCA \leftarrow$ child of $HCA$ who is an ancestor of $result\_class$
11. Until forever
```

Fig. 2. Pseudo code for recovery oriented error control

Fig. 2 shows the pseudo code of the recovery oriented scheme. Before we do the real classification, the CI of nodes l levels ahead are calculated so the list of classes that pass the CI test is known. The algorithm will select the optimal path with maximum local_prob(HCA, c) (defined below) among all such classes. If there is only one class passing the CI test, we jump to that class directly without further calculations. The functions used are listed below:

find_CI_list(c, l)	Suppose the level of c is i. Return a list of classes at level $l+i$ that passes CI test and any leaf classes between level $i+1$ and level $i+l-1$ that passes CI test.
local_prob(r_0, r_n)	Suppose r_0 is an ancestor of r_n and the path along r_0 to r_n is $r_0 \rightarrow r_1 \rightarrow \cdots \rightarrow r_n$. Return $p(r_1\|r_0)\ p(r_2\|r_1) \cdots p(r_n\|r_{n-1})$. In plain words, we are multiplying the one-step probabilities along r_0 to r_n together to get $p(r_n\|r_0)$.

3.2 Error Masking

The error masking scheme is based on the idea behind software fault tolerance. Instead of detecting error and then performing recovery, we use multiple programs employing different designs. Among the outputs generated by these pro-

```
Algorithm hc_error_mask($T$,$d$,$l$,$f$,$f'$)
// $T$: class taxonomy
// $d$: document to be classified
// $l$: difference of the level of HCA and that of the current node
// $f, f'$: two feature lengths for use with two O-classifiers
1.  $n \leftarrow$ root($T$) //level zero
2.  $level \leftarrow l + 1$
3.  $(c_1, c_2, c_3) \leftarrow (n, n, n)$
4.  While $c_1$ is not leaf do
5.      Start three threads:
6.          (i)     $c_1 \leftarrow$ O-classifier($c_1$, $T$, $d$, $level$, $f$)
7.          (ii)    $c_2 \leftarrow$ N-classifier($n$, $T$, $d$, $level$)
8.          (iii)   $c_3 \leftarrow$ O-classifier($c_1$, $T$, $d$, $level$, $f'$)
9.      Wait the finish of all threads
10.     If not ($c_1 = c_2 = c_3$) Then
11.         $c_1 \leftarrow$ majority of $c_1$, $c_2$ and $c_3$ (take a predefined action, e.g. using $c_2$, if no majority)
12.         $level \leftarrow level + 1$
13.         $n \leftarrow$ child of $n$ who is an ancestor of $c_1$
14.     Else
15.         $level \leftarrow level + l$
16.         $n \leftarrow c_1$
17.     Endif
18. Endwhile
19. Return $c_1$
```

Fig. 3. Pseudo code for error masking scheme

grams, the one generated by a majority is considered correct. More programs will generate more reliable results, but consume more resource.

We adopt a moderate approach. We run three classification methods in parallel. The first and third classifications are hierarchical classifications of traditional sense. The second classification is performed by dynamically skipping some levels in the class taxonomy. For example, to classify a document based on the taxonomy in Fig. 1, we can perform an additional classification by first skipping level 1 (i.e. {Recreation, Business}). Say the three classifications end up with class 'Dancing'. We then classify it at node 'Recreation'. But this time we skip the left part of level 2, i.e., {Sports, Dancing}. In the following discussion, we use the terms 'N-classifier' and 'O-classifier' respectively to refer to the classifiers with and without skipping the levels. The third classification is to employ O-classifier again but with a different feature length. A majority voting scheme is used to decide the overall output.

Skipping some levels has the effect of (partially) globalizing the information for the classification, and therefore can possibly reducing the misclassification rate. The more levels skipped, the more likely it is to reduce misclassification rate. In the extreme, if all but the leaf and root levels are skipped, we have a flat classifier. However, skipping a large number of levels beats one of the main motivations for using a hierarchical classifier, i.e., handling the complexity involved in the document classification. Thus a trade-off must be made. How to make such a trade-off is application dependent and is determined by users. In general, more levels can be skipped if the taxonomy has a large height but a small width than the other way around.

Fig. 3 shows the pseudo code for the error masking scheme. At line 11, if there is no majority formed by the classifiers, a user-defined action should be taken. For our experiments, this action is to use c_2, because usually, this is the most accurate (and slowest) classifier. Line 15-16 are some optimization codes. If c_1, c_2 and c_3 all match, we are confident that it is on the right track and so we can make a bold move — Instead of advancing one level, our algorithm moves l levels ahead. Some functions used are defined below:

O-classifier(c, T, d, k, f)	To classify the document d using O-classifier in the taxonomy T from class c to reach a class in level k or a leaf class at a level higher than k. The feature length to use in classification is f.
N-classifier(c, T, d, k)	To classify the document d using N-classifier in the taxonomy T from class c to reach a class in level k or a leaf class at a level higher than k.

4 Performance Evaluation

In this section, we study the performance of the algorithms. We implemented five document classification algorithms in C++. Our algorithms, namely the recovery oriented and the error masking schemes, are compared against simple hierarchical classification, flat classification and TAPER [4]. We run the experiments on a Sun Enterprise E4500 machine with 12 processors. Response time, rather than total CPU time, is measured so the error masking scheme can take advantage of parallelism.

We are interested in data sets with reasonably large class taxonomies, because the advantages of skipping levels can only be fully exploited in such data sets. We have chosen the data set of US patents[2] because they are organized in a large taxonomy. Three sets of data are collected from the US patent database. For convenience, we name them *Data_388*, *Data_TAPER* and *Data_Four*. Data_388 is the top-level class numbered 388 (motor control system) on the patent database. The class taxonomy is formed by all the 98 subclasses under the class 388. In each subclass, we download at most 20 patents, resulting in 901 patents. Data_TAPER highly resembles a data set used in [4]. The taxonomy of this data set is shown at Fig. 4(a). There are 12 leaf classes, each of which is a top-level class in the patent database. There are 500 training patents and 300 validation patents picked randomly from each leaf. However, since Data_TAPER is a three level data set, it is insufficient to demonstrate all the features of our algorithms while Data_388 only consists of a small number of patents. In Data_Four, we expand Data_TAPER by introducing more classes from the US patent database and grow the taxonomy by one level. The resulting class taxonomy is shown at Fig. 4(b).

Fig. 5, 6 and 7 show the accuracy and performance of the different algorithms on the three data sets. First of all, we achieve 65-70% accuracy in Data_TAPER,

[2] Available at several places on Internet, e.g. Delphion Intellectual Property Network (http://www.delphion.com/).

Patent:	Communication:	329 Demodulator
		332 Modulator
		343 Antenna
		379 Telephony
	Electricity:	307 Transmission
		318 Motive
		323 Regulator
		219 Heating
	Electronics:	330 Amplifier
		331 Oscillator
		338 Resistor
		361 System

(a) Class taxonomy for Data_TAPER

Patent:	Physics and Chemistry:	Physics:	131 Fluid handling
			261 Contact apparatus
			096 Gas separation appratus
			095 Gas separation process
		Chemistry:	422 Chemistry apparatus etc
			423 Inorganic compounds
			071 Fertilizers
			585 Hydrocarbon compounds
	Engineering:	Communication:	329 Demodulator
			332 Modulator
			343 Antenna
			379 Telephony
		Electricity:	307 Transmission
			318 Motive
			323 Regulator
			219 Heating
		Electronics:	330 Amplifier
			331 Oscillator
			338 Resistor
			361 System

(b) Class taxonomy for Data_Four

Fig. 4. Class taxonomies for some data sets used in the experiments

which is similar to the result in [4]. Among all the experiments, simple hierarchical classification is always the fastest algorithm and therefore it is the baseline of our comparison. Generally speaking, it is not justified to use a more complicated classification scheme unless it is more accurate than the fastest algorithm.

As simple hierarchical classification classifies the document in a greedy manner but TAPER searches the whole tree for the maximum overall probability, TAPER guarantees at least as good accuracy as simple hierarchical classification, although more time is required for the extra search. From the experiments, however, TAPER gives almost the same accuracy as the simple hierarchical classification. This result suggests the greedy approach of the simple hierarchical algorithm is close to optimal. Exhaustive search does not help to boost the accuracy. If we are to increase the accuracy, there must be a different approach to classify the documents. This is another reason to skip levels.

Flat classification gives the best accuracy in most cases[3]. However, from our experiments, it is clear that this is also the most time consuming algorithm except in the smallest taxonomy (Data_TAPER). Our algorithms stand on a middle ground between speed and accuracy. Our algorithms consistently beat TAPER and simple hierarchical algorithms in terms of accuracy. The recovery oriented scheme even slightly suppresses the flat classification in accuracy on Data_TAPER. However, it does not run fast since the recovery oriented scheme is essentially doing both a simple hierarchical and flat classification in a three level data set. In a bigger taxonomy (Data_388 and Data_Four), the recovery oriented scheme is clearly faster than flat classification, and at the same time more accurate than TAPER and simple hierarchical classification.

Like the recovery oriented scheme, the error masking scheme is also faster than flat classification and more accurate than TAPER and simple hierarchical classification in a large taxonomy. When comparing between the two error control schemes, it is found that there are some cases that either scheme is faster than the other. Due to parallelism, it is easy to understand why the error masking scheme is faster. However, the optimization adopted in our implementation also

[3] There seems to be no theoretical support for this in the general case. For example, the contrary is claimed in [4].

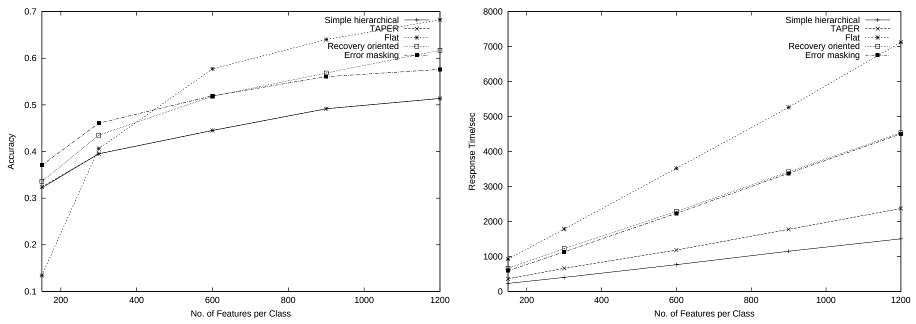

Fig. 5. Experimental result on Data_388

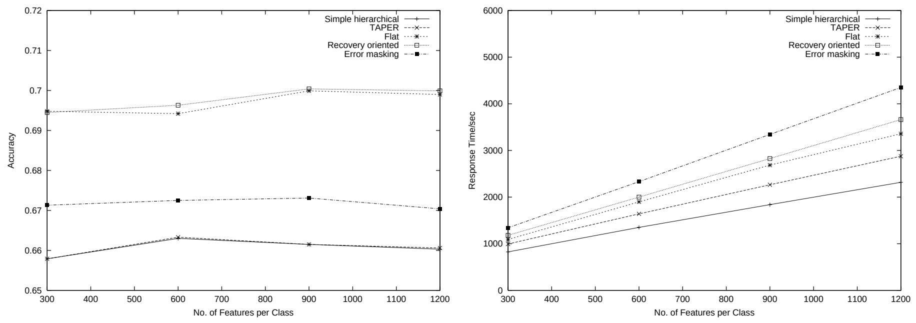

Fig. 6. Experimental result on Data_TAPER

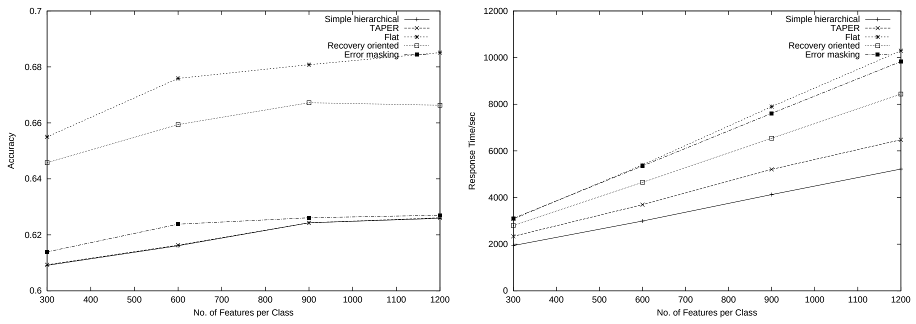

Fig. 7. Experimental result on Data_Four

gives the recovery oriented scheme an edge which explains why the recovery oriented scheme is faster in many cases. In recovery oriented scheme, if only one class is found to pass the closeness indicator, we will jump to that class directly. The one-step probability is not calculated. This is a considerable saving in running time. In contrast, the error masking scheme always classifies the document by three different classifiers and the slowest one will determine the response time. As for accuracy, the error masking scheme, while still ahead of TAPER and simple hierarchical classification, is often less accurate than the recovery oriented scheme. As the recover oriented scheme does not require a multi-processor machine, we feel that the recovery oriented scheme is preferable over the error masking scheme.

In a serious application, we expect a large class taxonomy. From the experiments, the response time difference between flat and simple hierarchical classification widens as the size of taxonomy grows. While the accuracy of simple hierarchical classification may not be satisfactory, switching to flat classification is too radical and computationally expensive. As our algorithms can be faster than flat classification at a taxonomy of as low as four levels (Data_Four), they represent a good trade-off between speed and accuracy for most applications.

5 Related Work and Conclusion

Classification has been studied extensively in the last decades [2,3,9,13,15,14, 17,21]. However, most of the work on the classification ignores the hierarchical structure of classes. In [1], the authors explore the hierarchical structure of attributes to improve the efficiency, but assume only a single level of classes. The work reported in [4,12] propose hierarchical classification based on the class taxonomy in the context of document classification. The work in [20] discusses document classification without using hierarchical classification. Bayesian network as a model for data mining has been studied in [6,5,10,7]. Feature selections are discussed in some work [11,16]. The general method is to define a measure first and then search for a subset of features that optimize this measure. Fisher Index method in [4] also follows this line, it does so however in a 'localized' manner, i.e. one term at a time. Although this local method has the weakness of not considering the fact that sometimes terms may be related, it does reduce the complexity when the number of features is very large.

In this paper, we have studied document classification using hierarchical classifiers with error control capability. We demonstrate that some well established strategies in other areas can also find a way to enhance the performance in our context. Two methods are proposed, recovery oriented and error masking. Recovery oriented method 'detects' an error and rejects it, while error masking method 'masks' an outcome under suspicion by adopting a better one. Our experiments show that both methods consistently reduce the misclassification rate against TAPER and simple hierarchical classification. The cost is extra running time, but they are faster than flat classification on a large taxonomy. Our algorithms are suitable for classifying documents into a large taxonomy where the users are willing to spend the extra time to trade for a higher accuracy.

References

1. H. Almualim, Y. Akiba, S. Kaneda, “An efficient algorithm for finding optimal gain-ratio multiple-split tests on hierarchical attributes in decision tree learning”, Proc. of National Conf. on Artificial Intelligence, AAAI 1996, pp 703 - 708.
2. R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer and A. Swami, “An interval classifier for database mining applications”, Proc. of VLDB, 1992, pp 560 - 573.
3. L. Breiman, J. Friedman, R. Olshen and C. Stone, “Classification and regression trees”, Wadsworth, Belmont, 1984.
4. S. Chakrabarti, B. Dom, R. Agrawal and P. Raghavan, “Using taxonomy, discriminants, and signatures for navigating in text databases”, Proc. of the 23rd VLDB, 1997, pp 446 - 455.
5. K. Cios, W. Pedrycz and r. Swiniarski, “Data mining methods for knowledge discovery”, Kluwer Academic Publishers, 1998.
6. P. Cheeseman, J. Kelly, M. Self, “AutoClass: a Bayesian classification system”, Proc. of 5th Int’l Conf. on Machine Learning, Morgan Kaufman, June 1988.
7. N. Friedman and M. Goldszmidt,“Building classifiers using Bayesian networks”, Proc. of AAAI, 1996, 1277 - 1284.
8. T. Fukuda, Y. Morimoto and S. Morishita, “Constructing efficient decision trees by using optimized numeric association rules”, Proc. Of VLDB, 1996, pp 146 - 155.
9. J. Gehrke, R. Ramakrishnan and V. Ganti, “Rainforest - a framework for fast decision tree construction of large datasets”, Proc. of VLDB, 1998, pp 416 -427.
10. D. Heckerman, “Bayesian networks for data mining”, *Data Mining and Knowledge Discovery*, 1, 1997, pp 79 - 119.
11. D. Koller and M. Sahami, “Toward optimal feature selection”, Proc. of Int’l. Conf. on Machine Learning, Vol. 13, Morgan-Kaufmann, 1996.
12. D. Koller and M. Sahami, “Hierarchically classifying documents using very few words”, Proc. of the 14th Int’l. Conf. on Machine Learning, 1997, pp 170 - 178.
13. M. Mehta, R. Agrawal and J Rissanen, “SLIQ: a fast scalable classifier for data mining”, Proc. of fifth Int’l Conf. on EDBT, March 1996
14. J. Quinlan, “Induction of decision trees”, Machine Learning, 1986, pp 81 - 106.
15. J. Quinlan, “C4.5: programs for machine learning”, Morgan Kaufman, 1993.
16. G. Salton, “Automatic text processing, the transformation analysis and retrieval of information by computer”, Addison - Wesley, 1989.
17. J. Shafer, R. Agrawal and M. Mehta, “Sprint: a scalable parallel classifier for data mining”, Proc. of the 22nd VLDB, 1996, pp 544 - 555.
18. E. S. Ristad, “A natural law of succession”, Research report CS-TR-495-95, Princeton University, July 1995.
19. S. Weiss, and C. Kulikowski, “Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning and expert systems”, Morgan Faufman, 1991.
20. K. Wang, S. Zhou and S. C. Liew, “Building hierarchical classifiers using class proximity”, Proc. of the 25th VLDB, 1999, pp 363 - 374.
21. Y. Morimoto, T. Fukuda, H. Matsuzawa, T. Tokuyama and K. Yoda, “Algorithms for mining association rules for binary segmentations of huge categorical databases”, Proc. of VLDB, 1998.