A decision-tree-based symbolic rule induction system for text categorization

by D. E. Johnson
F. J. Oles
T. Zhang
T. Goetz

We present a decision-tree-based symbolic rule induction system for categorizing text documents automatically. Our method for rule induction involves the novel combination of (1) a fast decision tree induction algorithm especially suited to text data and (2) a new method for converting a decision tree to a rule set that is simplified, but still logically equivalent to, the original tree. We report experimental results on the use of this system on some practical problems.

The text categorization problem is to determine predefined categories for an incoming unlabeled message or document containing text, based on information extracted from a training set of labeled messages or documents. Text categorization is an important practical problem for companies that wish to use computers to categorize incoming electronic mail, thereby either enabling an automatic machine response to the e-mail or simply assuring that the e-mail reaches the correct human recipient. But, beyond e-mail, text items to be categorized may come from many sources, including the output of voice recognition software, collections of documents (e.g., news stories, patents, or case summaries), and the contents of Web pages.

Previous text categorization methods have used decision trees (with or without boosting),[1] naive Bayes classifiers,[2] nearest-neighbor methods,[3] support vector machines,[4,5] and various kinds of direct symbolic rule induction.[6] Among all these methods, we are particularly interested in systems that can produce symbolic rules, because rules comprehensible to humans often provide valuable insights in many practical problems.

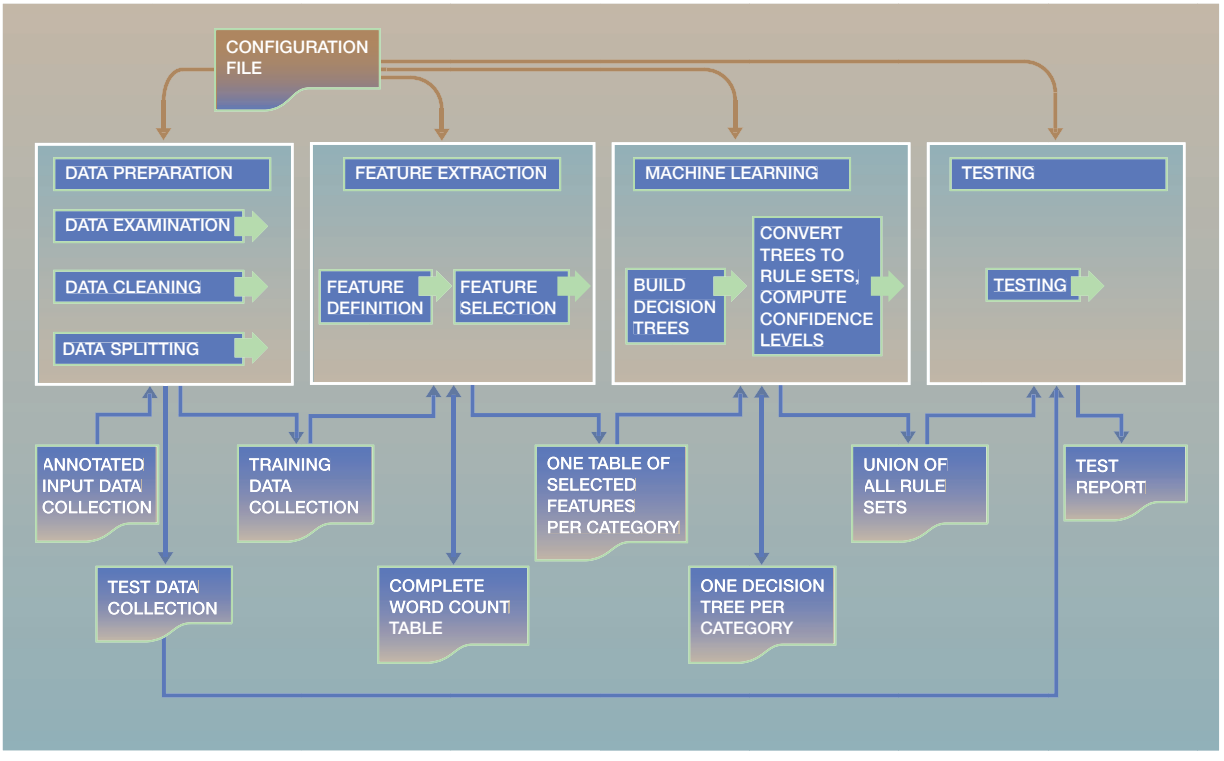
In a symbolic rule system, text is represented as a vector in which the components are the number of occurrences of a certain word in the text. The system induces rules from the training data, and the generated rules can then be used to categorize arbitrary data that are similar to the training data. Each rule ultimately produced by such a system states that a condition, which is usually a conjunction of simpler conditions, implies membership in a particular category. The condition forms the *antecedent* of the rule, and the conclusion posited as true when the condition is satisfied is the *consequent* of the rule. Usually, the antecedent of a rule is a combination of tests to be done on various components. For example:

```
share>3 & year<1 & acquire>2 → acq
```

may be read as "if the word *share* occurs more than three times in the document and the word *year* occurs at most one time in the document and the word *acquire* occurs more than twice in the document, then classify the document in the category *acq*."

In this paper, we describe a system that produces such symbolic rules from a decision tree. We emphasize the novel aspects of our work: a fast decision tree construction algorithm that takes advan-

Figure 1 The KitCat system architecture



tage of the sparsity of text data, and a rule simplification method that converts the decision tree into a logically equivalent rule set. Compared with a standard C4.5 tree algorithm,[7] our algorithm not only can be as much as a hundred times faster on some text data, but also can handle larger feature sets, which in turn leads to more accurate rules.

The prototype system described in this paper is available in product form as the IBM Text Analyzer, an IBM WebSphere* Business Component, and it is also embedded in the IBM Enterprise Information Portal, version 8, an IBM content-management product offering.

System architecture

All of the methods described in this paper have been implemented in a prototype text categorization system that we call *KitCat*, an abbreviation of "tool kit for text categorization." The overall KitCat system architecture is shown in Figure 1.

In order to support multiple categorization (a common requirement in text categorization applications), the KitCat system induces a separate rule set for each category by treating categorization *vis-a-vis* each category as a separate binary-classification problem. This has the secondary advantage that a person who analyzes the rules can then concentrate on one category at a time. Also, we associate with each rule a confidence measure that is the estimated in-class probability that a document satisfies the rule. In settings where it is desirable to assign a single category to a document, confidence levels enable the user to resolve the ambiguity resulting from the multiple firing of rules for different categories.

The decision tree induction algorithm

DTREE is our implementation of the decision tree induction algorithm, the component of KitCat used to carry out the "build decision trees" function in Figure 1. In this section we describe the novel features of the DTREE algorithm.

We start with a set of training documents indexed by consecutive integers. For each integer i in the index set, we transform the ith document into a numeric vector $x_i = (x_{i,1}, \ldots, x_{i,d})$ whose components are counts of the numbers of occurrences of words or features in the ith document. For each category, we would like to determine a label $y_i \in \{0, 1\}$ so that $y_i = 1$ indicates that the ith document belongs to the category, and $y_i = 0$ indicates that the ith document does not belong to the category.

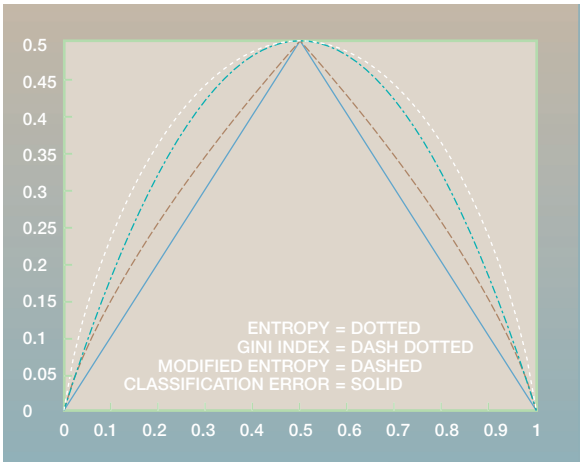
The dimension d of the input vector space is huge for text categorization problems. Usually a few hundred to a few thousand words (features) are required to achieve good performance, and tens of thousands of features may be present, depending both on the size of the data set and on the details of feature definition, such as tokenization and stemming. Despite the large number of features potentially present in text, the lengths of individual documents, especially for e-mail and Web applications, can be rather short on average. The result is that each data vector x is typically very sparse.

However, traditional tree construction algorithms (such as C4.5[7] or CART[8]) were not designed for sparse data. In the following, we briefly describe the fast decision tree algorithm that we developed specifically for text categorization applications. We assume that the reader has a basic knowledge of decision tree generation algorithms, so that we only elaborate on issues that are different from standard procedures, such as methods described by Quinlan[7] and Breiman et al.[8] For a discussion of various issues in developing scalable decision tree algorithms, see Gehrke et al.[9] While the construction of a tunable decision tree induction system especially designed for text categorization was mentioned in Yang et al.,[10] few details were given there, so we cannot compare our system to theirs.

Similar to a standard decision tree construction algorithm, our algorithm contains two phases: tree growing and tree smoothing. There are three major aspects of our algorithm that we describe in more detail: (1) taking advantage of the sparse structure, (2) using a modified entropy as the impurity measure, and (3) using smoothing to do pruning.

Tree growing. We employ a greedy algorithm to grow the tree. At each node corresponding to a subset T of the training data, we select a feature f and a value v so that data in T are partitioned into two subsets $T^1_{f,v}$ and $T^2_{f,v}$ based on whether or not $x_{i,f} \leq v$:

Figure 2 Functions that can be used to define impurity criteria



$T^1_{f,v} = \{x_i \in T : x_{i,f} \leq v\}$ and $T^2_{f,v} = \{x_i \in T : x_{i,f} > v\}$. This partition corresponds to a decision rule based on whether or not the word corresponding to feature f occurs more than v times in document d_i represented by vector x_i.

Let $p^1_{f,v} = P(y_i = 1 | x_i \in T^1_{f,v})$, $p^2_{f,v} = P(y_i = 1 | x_i \in T^2_{f,v})$, and $p_{f,v} = P(x_i \in T^1_{f,v} | x_i \in T)$. We consider the following transform of probability estimate $r : [0, 1] \rightarrow [0, 1]$ defined as

$$r(p) = \begin{cases} \frac{1}{2}(1 + \sqrt{2p - 1}) & \text{if } p > 0.5, \\ \frac{1}{2}(1 - \sqrt{1 - 2p}) & \text{if } p \leq 0.5. \end{cases} \quad (1)$$

We then define a modified entropy for $p \in [0, 1]$ as

$$g(p) = -r(p) \log (r(p)) - (1 - r(p)) \log (1 - r(p)). \quad (2)$$

For each possible split (f, v), we associate with it a cost function

$$Q(f, v) = p_{f,v} g(p^1_{f,v}) + (1 - p_{f,v}) g(p^2_{f,v}) \quad (3)$$

that measures the impurity of the split. The modified entropy criterion is plotted as a function of p in Figure 2, where we compare it with classification error, Gini index (we use $2p(1 - p)$ as its definition), and the standard entropy metric. Note that the

strict concavity of the modified entropy criterion indicates that it favors a split that enhances the purity of the partition, and, in this respect, it is similar to the Gini index and the entropy criterion.

Due to its flatness (nonstrict concavity), the classification error function does not favor a partition that enhances the purity. Consider an example such that $p_{f,v} = 0.5$, $p^1_{f,v} = 0.51$ and $p^2_{f,v} = 0.89$ and $p_{f',v'} = 1$, $p^1_{f',v'} = 0.7$ and $p^2_{f',v'} = 0$. The (f', v') partition is equivalent to no partition. Clearly, the classification error criterion gives equal scores for both partitions, since they both predict $y = 1$ for all data in T. However, in reality, (f, v) should be preferable, because it makes some progress: it is likely that we can further partition $T^1_{f,v}$ so that part of the data will be associated with a probability of $y = 1$ smaller than 0.51, and hence we change the prediction from 1 to 0 for those data. This future prospect of making a different prediction (increased purity of part of the data) is not reflected in the classification error criterion. This is why it is a bad measure for building a decision tree.

This problem can be remedied by using a strict concave, bell-shaped function. However, the widely adopted entropy and Gini index criteria have shapes too different from the classification error curve. The final criterion for evaluating a tree is by its classification performance; therefore we know that a tree with small entropy or Gini index does not automatically guarantee a small classification error. The modified entropy is introduced to balance the advantage of classification error, which reflects the final performance measurement of the tree, and the advantage of the entropy criterion, which tries to enhance the prospect of good future partitions within a greedy algorithm setting. It is designed to be a strictly concave function that closely resembles classification error. Note that, similar to the classification error function, the modified entropy curve is also nondifferentiable at $p = 0.5$. From our experience, this criterion outperforms the entropy and the Gini index criteria for text categorization problems in which we are interested. For example, on the standard Reuters-21578 data,[11] the new criterion reduces the overall number of errors by 6 percent compared with the entropy splitting criterion.

We now show how we can take advantage of the sparse structure presented in text documents to select the best partition. This aspect of our decision tree algorithm is crucial for its speed. Note that a standard decision tree algorithm such as CART or C4.5 does not take advantage of sparse structure. Since we are not aware of any previous work in the existing literature that carefully studied sparse decision tree methods, we describe this aspect of our algorithm in detail with analysis.

In order to speed up the algorithm, we also make an important modification to the data: that is, we truncate the word count $x_{i,f}$ to be at most a value V (typically, we choose this value to be 3). Note that this is a rather reasonable simplification for text categorization problems. For example, a ten-time occurrence of the word "car" is hardly a more informative predictor that a document is in a car-related category than a nine-time occurrence. Let d be the dimension of x, which indicates the number of features (words) under consideration. We create an array inclass-count$[1 \ldots d][0 \ldots V]$, where inclass-count$[f][v]$ is the number of documents $x_i \in T$ such that $y_i = 1$ and $x_{i,f} = v$; and an array total-count $[1 \ldots d][0 \ldots V]$, where total-count$[f][v]$ is the number of documents $x_i \in T$ such that $x_{i,f} = v$. The time required to create and fill the table is $O(|T| \cdot \bar{l}_T + dV)$, where $\bar{l}_T$ is the average nonzeros of $x_i \in T$. This can be achieved by going through all the nonzeros of vectors $x_i \in T$ and appropriately increasing the counts of the corresponding entries in the inclass-count and the total-count arrays.

After this step, we loop through $f = 1, \ldots, d$: for each f, we let v go from 0 to V. We accumulate the total number of $x_i \in T$ such that $x_{i,f} \leq v$, and the total number of $x_i \in T$ such that $y_i = 1$ and $x_{i,f} \leq v$. The probability $p_{f,v}$, $p^1_{f,v}$ and $p^2_{f,v}$ can then be estimated to compute the cost $Q(f, v)$ defined in (3). We keep the partition (f, v) that gives the minimum cost. This step requires $O(dV)$ operations.

For the tree growing stage, we recursively split the tree starting from the root node that contains all documents, until no progress can be made. Assuming the documents roughly have equal length $\bar{l}$, then the total time required to grow the tree is roughly $O(nh_l\bar{l} + dVM)$, where M is the number of nodes in the tree, n is the total number of documents, and h_l is the average depth of the tree per document. In our experience with text categorization, the dominating factor in $O(nh_l\bar{l} + dVM)$ is the first term $O(nh_l\bar{l})$. As a comparison, a dense tree growing algorithm will have complexity at least of $O(nh_ld)$, which is usually at least ten times slower.

Tree smoothing. Many algorithms for constructing a decision tree from data involve a two-stage pro-

cess: the first stage is to grow a tree as large as possible to fit the data, using procedures similar to what we have described in the previous section. After this stage, the tree typically "over-fits" the data in the sense that it may perform poorly on the test set. Therefore a second phase is required to prune the large tree so that the smaller tree gives more stable probability estimates, which lead to better performance on the test set.

In this section, we describe a procedure, which instead of pruning the fully grown tree, re-estimates the probability of every leaf node by averaging the probability estimates along the path leading from the root node to this leaf node. To achieve this, we borrow the "tree-weighting" idea from data compression.[12,13] If we use the tree for compression of the binary class indicator string y_i based on x_i, then the tree-weighting scheme guarantees that the re-estimated probability achieves a compression ratio not much worse than that of the best pruned tree.[11,12] Because we apply this method to the transformed probability estimate $r(p)$ rather than to p directly, this theoretical result can be interpreted nonrigorously as follows: by using the re-estimated probability, we can achieve an expected classification performance on the test set not much worse than that of the best pruned tree.

Note that in statistics, this technique is also called *shrinkage* because it shrinks the estimate from a node deeper in the tree (which has a lower bias but higher variance because there are fewer data points) toward estimates from nodes shallower in the tree (which have higher biases but lower variance). In Bayesian statistics and machine learning, this method is also called *model averaging*. See Carlin and Louis[14] for more discussions from the statistical point of view.

The basic idea of our algorithm can be described as follows: consider sibling nodes T_1 and T_2 with a common parent T. Let $p(T_1)$, $p(T_2)$ and $p(T)$ be the corresponding probability estimates. The local re-estimated probability is $w_T p(T) + (1 - w_T)p(T_1)$ for T_1 and $w_T p(T) + (1 - w_T)p(T_2)$ for T_2. The local weight w_T and an accompanying function $G(T)$ are computed by mutual recursion, based on the following formulas:

$$\frac{w_T}{1 - w_T} = \frac{c \cdot \exp(-|T| g(p(T)))}{\exp(-|T_1| G(T_1) - |T_2| G(T_2))},$$

$$G(T) = \begin{cases} g(p(T)) + \dfrac{1}{|T|} \log\left(\left(1 + \dfrac{1}{c}\right) w_T\right) & \text{if } w_T > 0.5, \\ \dfrac{|T_1|}{|T|} G(T_1) + \dfrac{|T_2|}{|T|} G(T_2) & \\ \qquad + \dfrac{1}{|T|} \log((1 + c)(1 - w_T)) & \text{otherwise.} \end{cases}$$

The parameter c is chosen *a priori* to reflect the Bayesian "cost" of a split. For a leaf node T, we define $G(T) = g(p(T))$ and $w_T = 1$. We choose $c = 16$ in our applications but the performance is not very sensitive to this parameter.

These formulas may seem a little mysterious at first. They are in fact derived from applying the Bayesian model averaging method locally at each node (note that we replace the probability p at a node with its transformed estimate $r(p)$). From the bottom up, we re-estimate the local probability at each node and feed it into a new probability estimate by shrinking toward its parent estimate. The quality of the re-estimate at each node T is measured by $G(T)$, which is updated at each step based on the information gathered from its children. Therefore based on $G(T)$, we can compute w_T—the local relative importance of the current node compared to its children. Due to the limitation of space, we shall skip the detailed derivations and theoretical consequences. Interested readers are referred to Willems et al.[12] and Zhang.[13]

After calculating weight w_T for each node recursively (the computation is done bottom-up), we compute the global re-estimated probability for each tree node from top-down. This step averages all the estimates $r(p)$ from the root node T_0 to a leaf node T_h down a path $T_0 \ldots T_h$, based on weight w_{T_i}. Note that weight w_T is only the local relative importance, which means that the global relative importance of a node T_k is $\tilde{w}_k = \Pi_{i<k}(1 - w_i)w_k$ along the path. By definition, $\sum_{i=1}^{h} \tilde{w}_i = 1$ for any path leading to a leaf. The following recursive procedure efficiently computes the global re-estimate of the children T_1 and T_2 at the parent node T:

$$\hat{w}_{T_i} = \hat{w}_T(1 - w_T) \tag{4}$$

$$\tilde{r}(T_i) = \tilde{r}(T) + \hat{w}_{T_i} w_{T_i} r(p(T_i)), \tag{5}$$

where $r(p(T))$ is the transformation using Equation 1 of the probability estimate $p(T)$ at node T. At the root node, we let $\hat{w} = 1$. After $\tilde{r}(T_h)$ has been computed for a leaf node T_h, we can use $r^{-1}(\tilde{r}(T_h)$ as its probability estimate. The label of T_i is 1 if $\tilde{r}(T_i) >$

0.5 and 0 otherwise. We prune the tree from the bottom up by checking whether two siblings give identical labels; if so, we remove them and use the label for their parent. This procedure continues until we cannot proceed. Typically, for a large fully grown tree from the first stage, the pruned tree can be as little as 10 percent of the unpruned tree.

The smoothing procedure consistently enhances the classification performance of the tree. The running time is $O(M)$ where M is the number of nodes in the unpruned tree. Therefore the total complexity of our decision tree construction algorithm is $O(nh_t\bar{l} + dVM)$. As mentioned before, in practice, we observe a complexity of $O(nh_t\bar{l})$.

Turning decision trees into symbolic rule sets

An integral part of our system writes the decision tree as an equivalent set of human interpretable rules. The value of converting decision trees into simplified logically equivalent symbolic rule sets, instead of employing a text categorization system based solely on decision trees, is threefold:

1. A system based solely on generating decision trees from training data is useless for categories for which there are no training data. However, it is not hard for an intelligent person to write logical rules to cover such a situation. Creating a decision tree by hand for the purpose of text categorization would be much harder, particularly for a person who is not mathematically sophisticated. We envision that handwritten rule sets could be seamlessly incorporated with machine-generated rule sets, either permanently or as a stopgap measure pending the collection of additional training data.

2. A human user can understand and modify a rule set much more easily than he or she can understand and modify a decision tree. There are a number of scenarios in which we envision the need for such modifications. For instance, there may be a discrepancy between the training data and the anticipated application that requires hand modification of an automatically created system, and, in a system such as the one we are describing, this modification may be accomplished by editing a rule file. Another scenario involves a user's desire to update an existing system in order to improve performance that may have degraded due to a changing environment (perhaps due to the introduction of new terminology or new products related to a category), without fully recreating the system from scratch.

3. The fact that a rule set is logically equivalent to a corresponding decision tree for a particular text categorization problem guarantees that any mathematical analysis of the overall performance of the decision tree (as opposed to the performance of individual rules) with respect to text categorization carries over to the rule set. This would not be true if a rule set only approximated a decision tree, as would be the case if the rule set were derived from the decision tree by heuristics.

The most straightforward way to convert a tree into an equivalent set of rules is to create a set with one rule for each leaf by forming the logical conjunction of the tests on the unique path from the root of the tree to the leaf. This starting point is laid out in detail by Quinlan[7] in the discussion of how C4.5 creates rule sets from trees. In C4.5, the initial rule set derived from a decision tree is modified based on the way changes in the rule set affect how the training data are classified. This can be a computational burden, but the burden is lessened by the use of plausible heuristics. However, one should note that the resulting rule set of C4.5 need not be logically equivalent to the rule set initially derived from the tree.

Our approach to deriving rule sets from decision trees differs markedly from that of C4.5. We want a fast algorithm that produces a simplified rule set that is *logically equivalent* to the rule set initially extracted from a decision tree. As long as the new rule set is logically equivalent to the original rule set, it does not need experimental validation to compare the new set's overall performance with that of the original set. We do not propose to get a provably minimal set of rules. Instead, our algorithm "picks the low-hanging fruit" by

1. Carrying out, within rules, all elementary logical simplifications related to greater than and less than, e.g., converting $(A < 3) \wedge (A < 2)$ to $(A < 2)$.
2. Removing tests that are logically superfluous in the context of the entire rule set, when those tests are readily identifiable from the structure of our decision trees.

The former simplification is standard, but the latter is not. It changes the meaning of rules associated with

a particular leaf of a decision tree, while preserving the overall meaning of the rule set.

Here is the precise algorithm that implements the second rule simplification. It works for a decision tree for a two-class problem (i.e., each leaf node is labeled by a class X or its complement Y), where our aim is to produce rules for membership in the class X.

For each leaf labeled X, create a rule for membership in the class X by forming a conjunction of conditions obtained by traversing the path from the root to X, but use those conditions only in conformance with the following:

> For each node N on a path from the root to a leaf labeled X, the condition attached to the *parent* of N is to be part of the conjunction only if the sibling criterion holds for N,
>
> where the sibling criterion for N is
>
> > The node N is not the root, and the sibling node of N is not a leaf node labeled by X.
>
> The resulting set of rules is then logically equivalent to the original tree.

It should be noted that the sibling criterion, and its use here, is novel.

For example, consider the decision tree shown in Figure 3 in which we assume each feature count is integer-valued, so that the negation of $A < 2$, for example, is $A \geq 2$. If one applies the algorithm to the decision tree in Figure 3, one sees that for each leaf labeled X, there is only one condition on the path from the root to the leaf that is not to be omitted from the conjunction, and so one immediately obtains the equivalent rule set

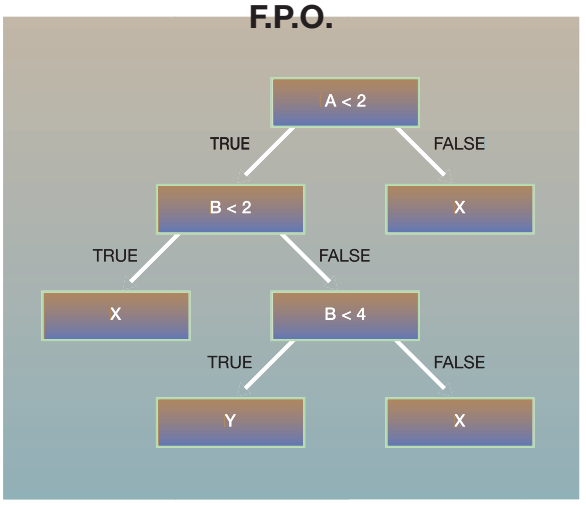
$$(B < 2) \rightarrow X$$

$$(B \geq 4) \rightarrow X$$

$$(A \geq 2) \rightarrow X$$

Why does this algorithm yield a set of rules equivalent to the standard set derived from a binary decision tree for a two-class problem? Each time a condition C is a candidate for elimination, there must be two leaf nodes labeled X, giving rise to two rules of the form

Figure 3 Example of a decision tree



$$P \wedge C \rightarrow X$$

$$P \wedge \neg C \wedge Q \rightarrow X$$

where P and Q are conjunctions of conditions. These two rules are equivalent to the single rule

$$(P \wedge C) \vee (P \wedge \neg C \wedge Q) \rightarrow X$$

However,

$$\begin{aligned}(P \wedge C) \vee (P \wedge \neg C \wedge Q) &= P \wedge (C \vee (\neg C \wedge Q)) = P \wedge (C \vee Q)) \\ &= (P \wedge C) \vee (P \wedge Q)\end{aligned}$$

Hence, the two rules above are logically equivalent to the two rules

$$P \wedge C \rightarrow X$$

$$P \wedge Q \rightarrow X$$

This verifies correctness of the algorithm.

In practice, decision trees obtained from text categorization problems are often very unbalanced, which may enable the procedure outlined to accomplish significant simplification.

Examples

One of the most common data sets used for comparing categorizers is the Reuters-21578 collection of categorized newswires.[11] We used the Mod-Apte data split, which gives a training set with 9603 items and a test set with 3299 items. Since these data are multiply classified, the performance is usually measured by precision and recall for each binary classification problem:

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \times 100$$

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \times 100$$

The overall performance can be measured by the micro-averaged precision, recall, and the break-even point computed from the overall confusion matrix defined as the sum of individual confusion matrices corresponding to the categories. Training on 93 categories, our results were micro-averaged precision of 87.0 percent and micro-averaged recall of 80.5 percent, the average of these two values being 83.8 percent. In this run, 500 features were selected by using the IG criterion in Yang and Pederson,[15] and no stopwords list was used. The training time for the DTREE decision tree induction algorithm on the Reuters-21578 data set was 80 seconds on a 400 megahertz Pentium II processor, so it is quite fast. (As a comparison, C4.5 takes hours to finish training on the same data.)

DTREE compares very favorably with results using the C4.5 decision tree induction algorithm, which has a micro-averaged precision/recall of 78.9 percent.[4] If one were willing to make the major sacrifice of giving up human-readable and human-modifiable rules, one could do better than DTREE alone by using boosting (i.e., adaptive resampling, with the accompanying generation of multiple trees). For example, boosted DTREE with 10 trees and 1000 features yields a precision of 89.0 percent and recall of 84.0 percent. With more than ten trees, additional improvement can also be obtained.

We have also used KitCat-generated human-comprehensible rules to study structures of corporate Web sites. In one instance, we obtained around 7000 Web pages using a Web crawler on the www.ibm.com Web site. These Web pages are partitioned into 42 categories based on the IBM Web site directory structure. Our goal was to analyze the Web site structure and see whether it would be possible to set up a good category scheme based on the existing directory structure. Such a category scheme could be used to help users to navigate through the Web site. To see how well KitCat works on these data, we studied its prediction accuracy under a three-fold cross-validation setting. We observed that the KitCat system worked very well on this Web site structure, with a prediction accuracy of about 86 percent. This compares favorably with our implementation of a naive Bayes method that gave an accuracy of about 75 percent, and a support vector machine (SVM) style classifier with an accuracy of about 87 percent. Note that the KitCat rule-based system has an added advantage of human interpretability. We also made some interesting observations in this study. For example, we found that the meta-text (for example, the *keywords* and the *description* sections) in an HTML (HyperText Markup Language) file were not as valuable as we originally expected—rules we obtained often did not contain these meta-data. In addition, removing them from the text representation does not have any significant impact on classification accuracy. In the following, we list a few example rules obtained from this data set:

```
room > 0 → press @ 0.79
room < 1 & press > 0 → press @ 0.16

webcasts > 0 & ibm > 0 & letter > 0 → news @
0.95
newswire > 0 & letter > 0 → news @ 0.9
newswire > 0 & letter < 1 → news @ 0.78

job > 0 → employment @ 0.87
resume > 0 → employment @ 0.91
employment > 0 → employment @ 0.6
```

For each rule, the word before "@" is the category name, and the number after "@" is the confidence measure (estimated probability). It is interesting to see that the word "room" is more indicative than "press" for the *press* category; "letter" is an indicative word for the *news* category; and both "job" and "resume" are more indicative than "employment" for the *employment* category. This shows that, in general, it can be difficult for a human to produce handwritten rules without sufficient knowledge of the data. On the other hand, machine-learned rules can give people valuable insights into a category scheme.

As another example, KitCat was also applied to categorize e-mail sent to a large U.S. bank. We had 4319

e-mail documents, with 3943 used for training and 970 used for testing. We used nine categories, covering 86 percent of the data. The rules produced by KitCat had a micro-averaged precision of 92.8 percent and a micro-averaged recall of 89.1 percent.

Our applications of KitCat have also taught us that it works very well on "dirty" data. Detailed analysis shows that the rules produced for classifying e-mail documents are quite good, even when the tester initially reports the contrary. On four categories in a customer data set, where we were surprised by less than stellar results as measured by the test data, KitCat rules actually were seen to have 100 percent precision, after the test data were carefully re-examined. The initial human categorization had precision ranging from 50.0 to 83.3 percent on these categories. There were 84 instances of these categories out of 8895 documents in the test set. Thus, misclassified test data can make even good rules look bad. However, bad data are a fact of life, and robustness in its presence is one of KitCat's advantages.

Conclusion

In this paper, we describe a decision-tree-based symbolic rule generation algorithm for text categorization. Our decision tree algorithm has three main characteristics:

1. Taking advantage of the sparse structure
2. Using a modified entropy as the impurity measure
3. Using smoothing to do pruning

We also have a procedure for turning a decision tree into a symbolic rule set by transforming a standard rule set into a simpler equivalent one based on an examination of the structure of the decision tree, and we give a proof of correctness for this procedure and some text categorization examples of the use of the resulting KitCat system. We created an industrial-strength state-of-the-art prototype system that eventually evolved into the IBM Text Analyzer product offering.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references

1. S. M. Weiss, C. Apte, F. Damerau, D. E. Johnson, F. J. Oles, T. Goetz, and T. Hampp, "Maximizing Text-Mining Performance," *IEEE Intelligent Systems* **14**, 63–69 (1999).
2. A. McCallum and K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification," *Proceedings*, *AAAI Workshop on Learning for Text Categorization*, Madison, WI (July 26–27, 1998), pp. 41–48.
3. Y. Yang, "An Evaluation of Statistical Approaches to Text Categorization," *Information Retrieval Journal* **1**, 69–90 (1999).
4. T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," *Proceedings*, *10th European Conference on Machine Learning*, Chemnitz, Germany (April 21–24, 1998), pp. 137–142.
5. S. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive Learning Algorithms and Representations for Text Categorization," *Proceedings*, *7th ACM International Conference on Information and Knowledge Management*, Washington, DC (November 3–7, 1988), pp. 148–155.
6. C. Apte, E. Damerau, and S. M. Weiss, "Automated Learning of Decision Rules for Text Categorization," *ACM Transactions on Information Systems* **12**, 233–251 (1994).
7. J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Francisco, CA (1993).
8. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth Advanced Books and Software, Belmont, CA (1984).
9. J. Gehrke, R. Ramakrishnan, and V. Ganti, "Rainforest—A Framework for Fast Decision Tree Construction of Large Datasets," *Data Mining and Knowledge Discovery* **12**, No. 2/3, 127–162 (2000).
10. Y. Yang, J. Carbonell, R. Brown, T. Pierce, B. Archibald, and X. Liu, "Learning Approaches for Detecting and Tracking News Events," *IEEE Intelligent Systems* **14**, No. 4, 32–43 (1999).
11. See http://www.research.att.com/~lewis.
12. F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The Context Tree Weighting Method: Basic Properties," *IEEE Transactions on Information Theory* **41**, No. 3, 653–664 (1995).
13. T. Zhang, "Compression by Model Combination," *Proceedings*, *IEEE Data Compression Conference*, Snowbird, Utah (March 30–April 1, 1998), pp. 319–328.
14. B. Carlin and T. Louis, *Bayes and Empirical Bayes Methods for Data Analysis*, Chapman and Hall, New York (1996).
15. Y. Yang and J. P. Pedersen, "A Comparative Study on Feature Selection in Text Categorization," *Proceedings*, *14th AAAI International Conference on Machine Learning*, Nashville, TN (July 8–12, 1997).

Accepted for publication March 26, 2002.

David E. Johnson *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York (electronic mail: dejohns@us.ibm.com).* Dr. Johnson is a research staff member at IBM's Watson Research Center, where he manages the computational linguistics and text mining group. His research interests include natural language processing, the syntax and semantics of natural language, and the foundations of linguistic theory. He holds a Ph.D. degree in theoretical linguistics from the University of Illinois at Urbana-Champaign, is a past president of the Association for the Mathematics of Language, and serves on several editorial boards.

Frank J. Oles *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York (electronic mail: oles@us.ibm.com).* Dr. Oles has been a research staff member at the IBM Watson Research Center since 1983. He has a Ph.D. degree in computer and information science from Syr-

acuse University. His interests include text categorization, information extraction from text, inductive learning of patterns, knowledge representation, the semantics of programming languages, and the mathematical foundations of computer science.

Tong Zhang *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York (electronic mail: tzhang@us.ibm.com).* Dr. Zhang received a B.A. degree in mathematics and computer science from Cornell University in 1994 and a Ph.D. degree in computer science from Stanford University in 1998. Since 1998, he has been with the IBM Watson Research Center, where he is now a research staff member in the Knowledge Management department. His research interests include machine learning, numerical algorithms, and their applications.

Thilo Goetz *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York (electronic mail: tgoets@us.ibm.com).* Dr. Goetz has been a research associate and later research staff member at IBM's Watson Research Center since 1997. He has a Ph.D. degree in computational linguistics from the University of Tübingen, Germany. His areas of interest include the intersection of computer science and computational linguistics, such as parsing, finite state methods, and compiler construction for computational linguistics, as well as complexity theory and logic for computer science.