Paper SAS1716-2015

Using Boolean Rule Extraction for Taxonomic Text Categorization for Big Data

Zheng Zhao, James Cox, Russell Albright, and Ning Jin
SAS Institute Inc.

ABSTRACT

Categorization hierarchies are ubiquitous in big data. Examples include MEDLINE's Medical Subject Headings (MeSH) taxonomy, United Nations Standard Products and Services Code (UNSPSC) product codes, and the Medical Dictionary for Regulatory Activities (MedDRA) hierarchy for adverse reaction coding. A key issue is that in most taxonomies the probability of any particular example being in a category is very small at lower levels of the hierarchy. Blindly applying a standard categorization model is likely to perform poorly if this fact is not taken into consideration.

This paper introduces a novel technique for text categorization, called Boolean rule extraction, which enables you to effectively address this situation. In addition, models that are generated by this introduced rule-based technique can be easily interpreted and modified by a human expert, enabling better human-machine interaction. The Text Rule Builder node and the newly developed HPBOOLRULE procedure in SAS® Text Miner implement this technique. The paper demonstrates how to use the HPBOOLRULE procedure to obtain effective predictive models at various hierarchy levels in a taxonomy.

INTRODUCTION

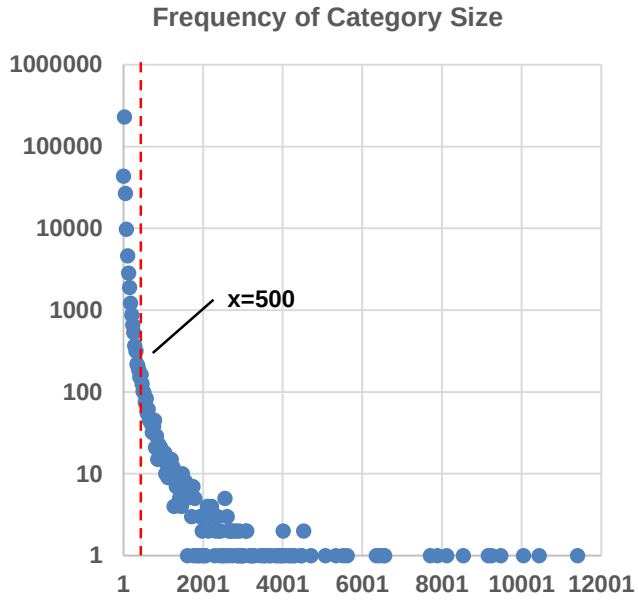
Hierarchies are becoming ever more popular for indexing and organizing large text corpora that contain millions or even billions of documents. Well-designed hierarchies are important for enterprise-level content categorization. The prevalence of complicated large-scale hierarchical taxonomies has generated a pressing need for the development of automated hierarchical text categorization. Most state-of-the-art hierarchical text categorization techniques fall into two classes of models: the top-down model and the flat model. A top-down model builds a classifier for each node of the hierarchy, and a document is classified as a member of a leaf category in the hierarchy if and only if it is also classified as a member of all the ancestor categories of the leaf category.[1] On the other hand, a flat model constructs a binary categorization problem[2] for each leaf category of the hierarchy, and a document is classified as a member of the leaf category if the prediction of the classifier that is built for the corresponding binary categorization problem is positive.

A flat model enjoys certain advantages over a hierarchical model. First, implementation of a flat model is simple, and most existing classifiers can be used directly as base building blocks. Second, despite its simplicity, a flat model might generate more accurate results in practice because it does not suffer the error propagation problem of a hierarchical model. When a document is classified wrongly on an upper level of a hierarchical model, the error is propagated to its descendant categories.

However, a flat model faces training difficulties. First, because it does not use a divide-and-conquer approach to solve a problem as a hierarchical model does, each of its base classifiers is usually exposed to the full text corpus. Being exposed to the full text corpus requires the base classifier to be highly efficient, especially when the size of the text corpus is large. Second, the base classifiers of the flat model usually face the issue of data sparsity: many taxonomies consist of a very large number of leaf categories, and the number of documents in a leaf category can be quite small even in a huge corpus. Because the binary categorization problem is usually constructed by considering all documents that belong to the category as positive and all documents that do not belong to the category as negative, there is a very high imbalance between positive and negative samples for the constructed problem. This issue requires the base classifier

[1] A leaf category is a node that has no children.
[2] Hierarchy information can still be incorporated by constructing the binary categorization problem in certain ways.

to be very effective for handling rare targets. Figure 1 presents information about the frequency and distribution of the category size for the Wikipedia data set,[3] which contains 2,365,432 documents organized under a hierarchy that has 325,056 non-empty categories. Figure 1 shows that about 90% of the categories in the hierarchy contain fewer than 50 documents (less than 0.002% of the data).



Distribution of Category Size

Size	Frequency	Percentage
0–10	202,777	62.38
11–50	95,086	29.25
51–100	15,447	4.75
101–500	10,618	3.27
501–1000	763	0.23
1001–5000	343	0.11
5001–10000	15	0.00
More	7	0.00

Number of samples: **2,365,436**

Number of category: **325,056**

Figure 1. Frequency and Distribution of Category Size of the Wikipedia Data

This paper introduces a novel technique for hierarchical text categorization via Boolean rule extraction. This technique can automatically generate a set of Boolean rules by analyzing a text corpus. For example, you could use this technique to generate the following rule set for documents that are related to bank interest:

(cut ^ rate ^ bank ^ percent ^ ¬ sell) *or*

(market ^ money ^ ¬ year ^ percent ^ ¬ sale) *or*

(repurchase ^ fee) *or*

(rate ^ prime rate) *or*

(federal ^ rate ^ maturity)

In this example, ^ indicates a logical *and*, and ¬ indicates a logical *negation*. The first line of the rule set says that if a document contains the terms “cut,” “rate,” “bank,” and “percent” but does not contain the term “sell,” it belongs to the bank interest category.

This technique has three advantages when you compare it to other techniques for hierarchical categorization. First, the technique focuses more on modeling the positive documents in a category; therefore, it is more robust when the data are imbalanced. Second, the rules can be easily interpreted and modified by a human expert, enabling better human-machine interaction. Third, the algorithm adopts a set of effective heuristics to significantly shrink the search space. These three advantages suggest that the technique can be applied as the base classifier for the flat model without incurring some of the disadvantages mentioned earlier. The basic operations of the algorithm are set operations, which are very fast. Therefore, the algorithm is highly efficient and can handle very-large-scale problems.

This paper also describes the new HPBOOLRULE procedure and the Text Rule Builder node in SAS Text Miner, which implement the introduced technique. The procedure provides a programming interface, and

[3] http://lshtc.iit.demokritos.gr/files/second/datasets/wikipediaLarge2.0.tar.gz

the Text Rule Builder node provides a GUI interface. Finally, the paper demonstrates how to use the HPBOOLRULE procedure to handle text categorization problems that have a complicated taxonomy.

BOOLEAN RULE EXTRACTION FOR TEXT DATA

Rule-based text categorization algorithms uses text rules to classify documents. Text rules are interpretable and can be effectively learned even when the number of positive document is very limited. BOOLLEAR (Cox and Zheng 2014) is a novel technique for Boolean rule extraction. When you supply a text corpus that contains multiple categories, BOOLLEAR extracts a set of binary rules from each category and represents each rule in the form of a conjunction, where each item in the conjunction denotes the presence or absence of a particular term. The BOOLLEAR process is as follows (criteria and measurements that are used in this process are described in the next section):

1. Use an information gain criterion to form an ordered term candidate list. The term that best predicts the category is first on the list, and so on. Terms that do not have a significant relationship to the category are removed from this list. Set the current term to the first term.
2. Determine the "estimated precision" of the current term. The estimated precision is the projected percentage of the term's occurrence with the category in out-of-sample data, using additive smoothing (Manning, Raghavan, and Schütze 2008). Create a rule that consists of that term.
3. If the "estimated precision" of the current rule could not possibly be improved by adding more terms as qualifiers, then go to step 6.
4. Starting with the next term on the list, determine whether the conjunction of the current rule with that term (via either term presence or term absence) significantly improves the information gain and also improves estimated precision.
5. If there is at least one combination that meets the criterion in step 4, choose the combination that yields the best estimated precision, and go to step 3 with that combination. Otherwise, continue to step 6.
6. If the best rule obtained in step 3 has a higher estimated precision than the current "highest precision" rule, replace the current rule with the new rule.
7. Increment the current term to the next term on the ordered candidate term list and go to step 2. Continue repeating until all terms in the list have been considered.
8. Determine whether the harmonic mean of precision and recall (the F1 score) of the current rule set is improved by adding the best rule obtained by steps 1 to 7. If it is not, then exit.
9. If so, remove all documents that match the new rule from the document set, add this rule to the rule set, and go to step 1 to start creating the next rule in the rule set.

MEASUREMENTS USED IN BOOLLEAR

This section provides detailed information about the measurements that are used in BOOLLEAR to evaluate terms and rules.

Precision, Recall, and the F1 Score

Precision measures the probability that the observation is actually positive when a classifier predicts it to be positive, recall measures the probability that a positive observation will be recognized, and the F1 score is the harmonic mean of precision and recall. A good classifier should be able to achieve both high precision and high recall. The precision, recall, and F1 score are defined as

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

where TP is the true-positive (the number of documents that are predicted to be positive and are actually positive), FP is the false-positive (the number of documents that are predicted to be positive but are actually negative), TN is the true-negative (the number of documents that are predicted to be negative and are actually negative), and FN is the false-negative (the number of documents that are predicted to be negative but are actually positive).

A classifier thus obtains a high F1 score if and only if it can achieve both high precision and high recall. The F1 score is a better measurement than accuracy[4] when the data are imbalanced, because a classifier can obtain very high accuracy by predicting that all samples belong to the majority category.

Information Gain Criterion

In steps 1 and 4, BOOLLEAR uses the *g*-test (Soka and Rohlf 1994), which is also known as the likelihood-ratio or maximum likelihood statistical significance test, as a form of information gain criterion to evaluate the correlation between terms and the target. The *g*-test generates by a *g*-score, which has two beneficial properties: As a form of mutual information, it is approximately equivalent to information gain in the binary case. And because it is distributed as a chi-square, it can also be used for statistical significance testing. The *g*-test is designed to compare the independence of two categorical variables. Its null hypothesis is that the proportions at one variable are the same for different values of the second variable. Given the TP, FP, FN, and TN of a term, its *g*-score can be computed as

$$g = 2\sum_{i=\{TP,TN,FP,FN\}} O(i)\log\left(\frac{O(i)}{E(i)}\right)$$

$$O(TP) = TP;\ O(FP) = FP; O(TN) = TN;\ O(FN) = FN$$

$$E(TP) = (TP + FP) * \frac{P}{P+N}$$

$$E(FP) = (TP + FP) * \frac{N}{P+N}$$

$$E(TN) = (TN + FN) * \frac{N}{P+N}$$

$$E(FN) = (TN + FN) * \frac{P}{P+N}$$

where P is the number of positive documents; N is the number of negative documents; O(TP), O(FP), O(TN), and O(FN) refer to the observed TP, FP, TN, and FN of a term; and E(TP), E(FP), E(TN), and E(FN) refer to the expected TP, FP, TN, and FN of a term. A term has a high *g*-score if it appears often in positive documents but rarely in negative documents, or vice versa.

Estimated Precision

Estimated precision helps BOOLLEAR shorten its search path and avoid generating overly specific rules. Computing the estimated precision is done by a form of additive smoothing with additional correction (err_i) to favor shorter rules over longer rules:

$$precision_i^m(t) = \frac{TP_{i,t} + \frac{P}{N+P} * m}{TP_{i,t} + FP_{i,t} + m} - err_{i-1}$$

$$err_i = \frac{TP_{i,t}}{TP_{i,t} + FP_{i,t}} - \frac{TP_{i,t} + \frac{P}{N+P} * m}{TP_{i,t} + FP_{i,t} + m} + err_{i-1}$$

In the preceding equations, the user-specified parameter m (≥1) is for bias correction. A large m is called for when a very large number of rules are evaluated, in order to minimize selection bias. $TP_{i,t}$ and $FP_{i,t}$ are the true-positive and false-positive of rule *t* when the length of the rule is *i*.

[4] Accuracy=(TP+TN)/(TP+FP+TN+FN)

Improvability Test

In step 3, BOOLLEAR conducts an improvability test for "in-process" model pruning. To determine whether a rule can be improved, BOOLLEAR applies the *g*-test on a perfect confusion table, which is defined as:

TP	**0**
0	**FP**

In the table, TP is the true-positive of the rule and FP is the false-positive of the rule. The *g*-score that is computed by using this table reflects the maximum *g*-score that a rule could possibly obtain if a perfectly discriminating term is added to the rule. If the *g*-score is smaller than a user-specified number that indicates a maximum *p*-value for significance, BOOLLEAR considers the rule to be unimprovable.

SHRINKING THE SEARCH SPACE

Exhaustively searching the space of possible rules is impractical because of the exponential number of rules that would have to be searched (2^m rules, where m is the number of candidate terms). Secondly, an exhaustive search usually leads to overfitting by generating many overly specific rules. Therefore, BOOLLEAR implements the following strategies to dramatically shrink the search space to improve its efficiency and help it avoid overfitting:

- **Feature selection**: BOOLLEAR uses the *g*-test to evaluate terms, and it adds only the top *n* terms to the ordered candidate term list for rule extraction. The size of the candidate term list controls the size of the search space. The smaller the size, the fewer the terms that are used for rule extraction, and therefore the smaller the search space.
- **Significance testing***:* In many rule extraction algorithms, rules are built until they perform perfectly on a training set, and pruning is applied afterwards. BOOLLEAR prunes "in-process." The following three checks are a form of in-process pruning, in that rules are not expanded when their expansion doesn't meet these basic requirements. These requirements help BOOLLEAR truncate its search path and avoid generating overly specific rules:
  - *Minimum positive document coverage*: BOOLLEAR requires that a rule be satisfied by at least *s* positive documents.
  - *Early stop based on the* g*-test*: BOOLLEAR stops searching when the *g*-score that is calculated for improving (or starting) a rule does not meet required statistical significance levels.
  - *Early stop based on estimated precision*: BOOLLEAR stops growing a rule when the estimated precision of the rule does not improve when the current best term is added to the rule. This strategy helps BOOLLEAR shorten its search path.
- ***k*-best search**: In the worst case, BOOLLEAR could still examine an exponential number of rules, although the heuristics described here minimize that chance. But because the terms are ordered by predictiveness of the category beforehand, a *k*-best search is used to further improve the efficiency of BOOLLEAR: If BOOLLEAR tries unsuccessfully to expand (or start) a rule numerous times with the a priori "best" candidates, then the search can be prematurely ended. Thus, there are two optional parameters, k_1 and k_2, that determine the maximum number of terms to examine for improvement. The first parameter can be added to step 7; if k_1 consecutive terms have been checked for building possible rules and none of them are superior to the best current rule, the search is terminated. Furthermore, in step 4, if k_2 consecutive terms have been checked to add to a rule and they don't generate a better rule, then the search for expanding that rule can be terminated. This helps BOOLLEAR shorten its search path (even with a very large number of candidate terms) with very little sacrifice to accuracy.
- **Improvability test**: This tests whether adding a theoretically perfect discriminating term to a particular rule could possibly both result in a statistically significant result and have a higher estimated precision than the current rule. If not, then the current rule is recognized without additional testing as the best possible rule, and no further expansion is needed.

- **Early stop based on the F1 score**: BOOLLEAR stops growing the rule set if adding the current best rule does not improve its F1 score. Thus, the F1 score is treated as the objective to maximize.

HPBOOLRULE PROCEDURE

The HPBOOLRULE procedure implements the BOOLLEAR technique and will be available in single-machine mode in SAS Text Miner 14.1. In single-machine mode, PROC HPBOOLRULE can use all the cores of a machine for parallel processing through multithreading. Distributed mode (which supports using multiple machines) will be supported in a future release; it will require a license for SAS® High-Performance Text Mining.

The following example shows how to first use the HPTMINE procedure to parse a text data set and then use its outputs in the HPBOOLRULE procedure for rule extraction:

```
proc hptmine
    data        =   docs;
doc_id              docid;
var                 text;
parse
    outparent   =   bow
    outterms    =   terms;
performance details;
run;

proc hpboolrule
    data        =   bow
    docid       =  _document_
    termid      =  _termnum_
    docinfo     =   docs
    terminfo    =   terms;
docinfo
    id          =   key
    targettype  =   binary
    targets     =   (target1 target2 target3);
terminfo
    id          =   key
    label       =   term;
output
    rules       =   rules
    ruleterms   =   ruleterms;
performance details;
run;
```

In the PROC HPTMINE step, the OUTPARENT= and OUTTERMS= options in the PARSE statement specify the names of data sets to contain, respectively, a term-by-document matrix (the data set Bow) and summary information about the terms in the document collection (the data set Terms).

In the PROC HPBOOLRULE statement, the Bow and Terms data sets are specified as input in the DATA= and TERMINFO= options, respectively. In addition, the DOCID= and TERMID= options in the PROC HPBOOLRULE statement specify the columns of the Bow data set that contain the document ID and term ID, respectively. The DOCINFO= option in the PROC HPBOOLRULE statement specifies the data set (Docs) to contain information about the documents, and the TERMINFO= option specifies the data set (Terms) to contain information about the terms.

The DOCINFO statement in the PROC HPBOOLRULE step specifies the following information about the Docs data set:

- The ID= option specifies the column that contains the document ID. The variables in this column are matched to the document ID variable that is specified in the DOCID= option in the PROC HPBOOLRULE statement to fetch target information about documents for rule extraction.
- The TARGETTYPE= option specifies the type of the target variables.

- The TARGETS= option specifies the target variables.

The TERMINFO statement specifies the following information about the Terms data set:

- The ID= option specifies the column that contains the term ID. The variables in this column are matched to the term ID variable that is specified in the TERMID= option in the PROC HPBOOLRULE statement to fetch information about terms for rule extraction.
- The LABEL= option specifies the column that contains the text of the terms.

Table 1 shows additional options in the PROC HPBOOLRULE statement that you can use to tune the procedure's performance.

Option	Default Value	Description
MPOS	8	*m* value for estimated precision computation
GPOS	8	Minimum *g-score* needed for the term or rule to be considered. This corresponds to $p < 0.004$.
MAXTRIESOUT	50	k_1 value for the *k*-best search for creating a rule that begins with a term
MAXTRIESIN	150	k_2 value for the *k*-best search to add a term to a rule
MINSUPPORTS	3	*s* value for minimum document coverage
MAXCANDIDATES	500	*n* value for the maximum number of terms to be selected as candidate terms

Table 1. PROC HPBOOLRULE Statement Options for Performance Tuning

TEXT RULE BUILDER NODE

SAS Text Miner is an optional add-on for SAS® Enterprise Miner™. The Text Rule Builder node, which can be accessed on the Text Mining tab of SAS Enterprise Miner, is a standard modeling node and has complete standard reporting features. The node enables you to develop repeatable and shareable process flows for predictive modeling of your text data.

TEXT RULE BUILDER NODE PROPERTIES

Figure 4 shows the properties of the Text Rule Builder node. Starting with SAS Text Miner 14.1, the node uses the HPBOOLRULE procedure to extract rules. In prior releases, the node used SAS macros and DATA steps to extract rules.

The following settings are specific to the Text Rule Builder node:

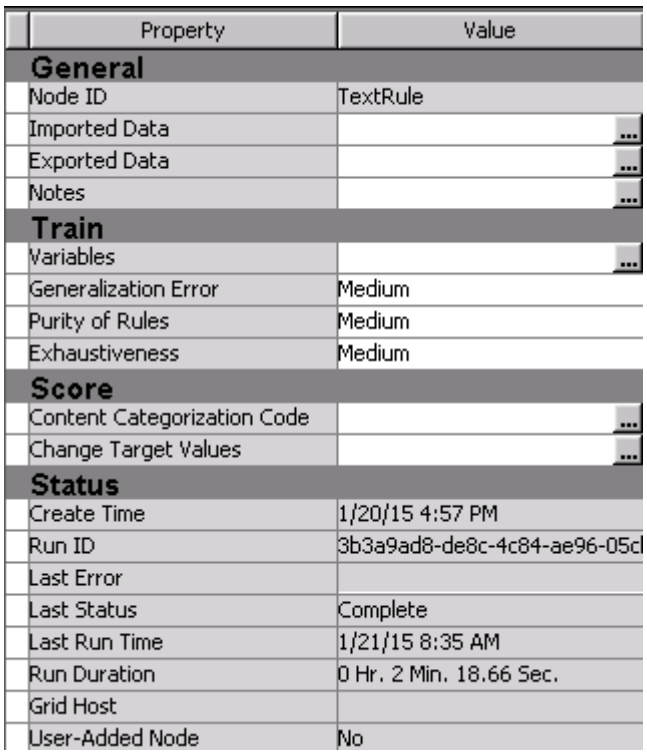
Generalization Error determines which *m* value is used in the precision *m*-estimate for preventing overfitting. Higher values do a better job of preventing overfitting at a cost of not finding potentially useful rules.

Purity of Rules determines which *g* value is used to test whether the rule can be improved. Higher values result in earlier termination of rule creation and less chance that bogus rules will enter in, but they also result in missing some marginally useful rules.

Exhaustiveness determines how many potential rules the rule search process considers at each step. This option controls the value of the k_1 and k_2 parameters that are used in the *k*-best search. As the exhaustiveness increases, the amount of processing time that the Text Rule Builder node requires also increases.

Content Categorization Code enables you to view the Content Categorization Code window. You can copy the code that is provided in that window and paste it into SAS® Content Categorization Studio.

Change Target Values enables you to view the Change Target Values window, where you can view and reassign target values. As a result, you can rerun the node and iteratively refine your model.



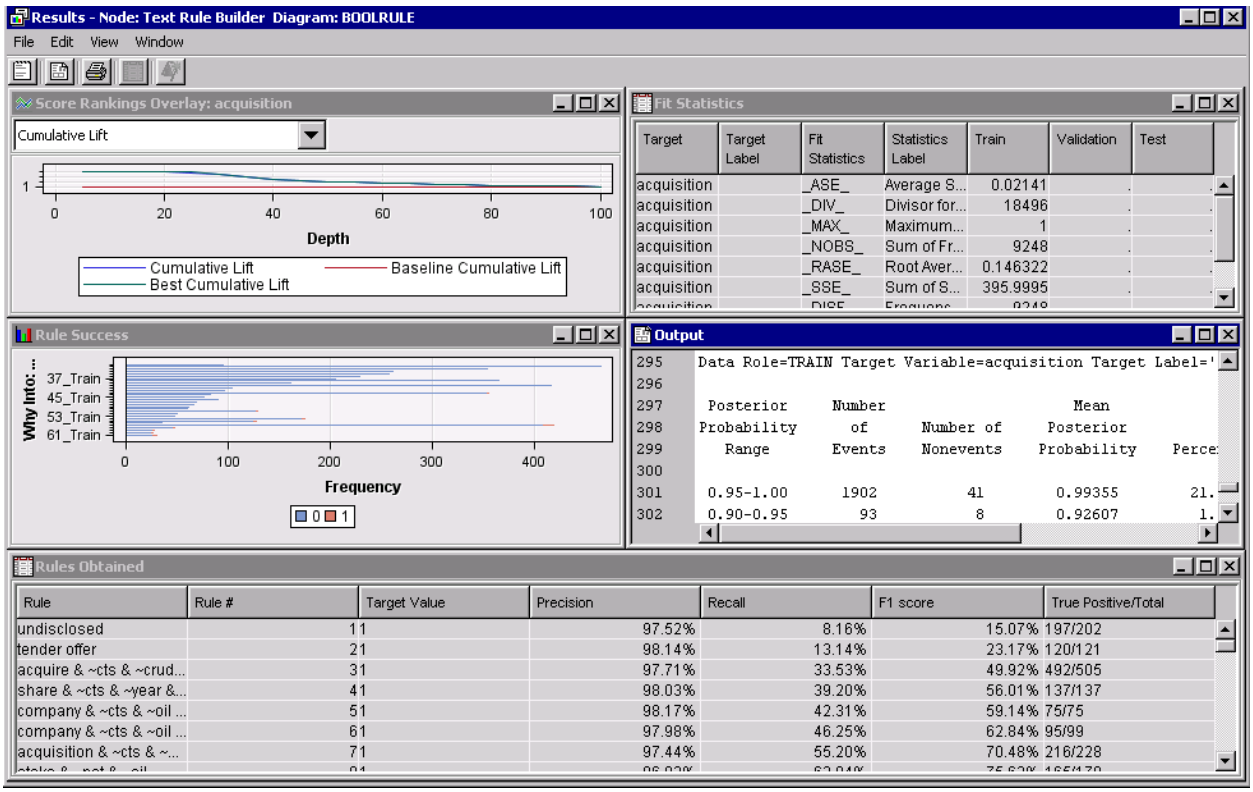
Property	Value
General	
Node ID	TextRule
Imported Data	
Exported Data	
Notes	
Train	
Variables	
Generalization Error	Medium
Purity of Rules	Medium
Exhaustiveness	Medium
Score	
Content Categorization Code	
Change Target Values	
Status	
Create Time	1/20/15 4:57 PM
Run ID	3b3a9ad8-de8c-4c84-ae96-05cl
Last Error	
Last Status	Complete
Last Run Time	1/21/15 8:35 AM
Run Duration	0 Hr. 2 Min. 18.66 Sec.
Grid Host	
User-Added Node	No

Figure 2. Text Rule Builder Properties

RESULT DIAGRAMS OF TEXT RULE BUILDER NODE

Figure 3**Error! Reference source not found.** shows the result diagrams that are displayed by the Text Rule Builder node. The following diagrams are specific to this node:

- The **Rules Obtained** table displays the discovered rules for predicting the target levels of the target variable. The order in which the rules are presented in this table is important and helps in the interpretation of the results. The first rule, “undisclosed,” was trained by using all of the data. The second rule, “tender offer,” was trained on the data that did not satisfy the first rule. Subsequent rules, such as “acquire & ~cts & ~crude,” are learned only after removing any observations that satisfy previous rules. The fit statistics to the right of each rule are cumulative and represent totals that include using that particular rule along with all of the previous rules in the table.
- The **Rule Success** graph shows the effectiveness of each generated rule on the training and validation data. In this graph, the rules on the Y axis are labeled with their rule number and the partition type. Within the partition, the rules are presented in the order in which they appear in the **Rules Obtained** table. The stacked bar to the side of each rule indicates the distribution of the target levels for the set of documents that satisfy the rule. Each bar shows you how effective the particular rule is and provides a visual indication of the correct target level for the false positives that the rule has incorrectly categorized.



Target	Target Label	Fit Statistics	Statistics Label	Train	Validation	Test
acquisition		_ASE_	Average S...	0.02141	.	.
acquisition		_DIV_	Divisor for...	18496	.	.
acquisition		_MAX_	Maximum...	1	.	.
acquisition		_NOBS_	Sum of Fr...	9248	.	.
acquisition		_RASE_	Root Aver...	0.146322	.	.
acquisition		_SSE_	Sum of S...	395.9995	.	.

```
Data Role=TRAIN Target Variable=acquisition Target Label='

 Posterior      Number                      Mean
Probability       of       Number of      Posterior
   Range        Events     Nonevents      Probability     Perce

0.95-1.00        1902          41          0.99355        21.
0.90-0.95          93           8          0.92607        1.
```

Rule	Rule #	Target Value	Precision	Recall	F1 score	True Positive/Total
undisclosed	11		97.52%	8.16%	15.07%	197/202
tender offer	21		98.14%	13.14%	23.17%	120/121
acquire & ~cts & ~crud...	31		97.71%	33.53%	49.92%	492/505
share & ~cts & ~year &...	41		98.03%	39.20%	56.01%	137/137
company & ~cts & ~oil ...	51		98.17%	42.31%	59.14%	75/75
company & ~cts & ~oil ...	61		97.98%	46.25%	62.84%	95/99
acquisition & ~cts & ~...	71		97.44%	55.20%	70.48%	216/228

Figure 3. Result Diagrams of Text Rule Builder Node

PERFORMANCE EXAMPLE

The HPBOOLRULE procedure implements very efficient data structures to handle sparse data and can support multithreading for parallel processing when your machine has multiple cores. You can use the procedure to process very large data sets even in single-machine mode. The following example examines the performance when the HPBOOLRULE procedure is used as the base classifier in the flat model.

EVALUATION DATA AND EXPERIMENT SETUP

The Wikipedia-Large data set[5] is derived from the web pages on Wikipedia. It contains 2,365,432 documents and 2,085,167 terms. The documents are organized under a hierarchy that has 325,056 non-empty categories. To evaluate the performance of BOOLLEAR when data are highly imbalanced, two sets of problems are extracted from the Wikipedia-Large data set. The first problem set is named WIKI50 and contains 10 problems, each of which has 50 positive documents and 2,365,382 negative documents. The data of each problem are further split into training, validation, and test data sets. Each of the validation and test data sets has 5 positive documents and 50 negative documents, and the remaining documents are put in the training data set. The second problem set is named WIKI500 and also contains 10 problems, each of which has 500 positive documents and 2,364,932 negative documents. The data of each problem are further split into training, validation, and test data sets. Each of the validation and the test data sets has 50 positive documents and 500 negative documents, and the remaining documents are put in the training data set. In each problem set, documents are represented as a sparse vector of 2,085,167 dimensions. Therefore, the whole corpus forms a 2,365,432 × 2,085,167 sparse matrix and is stored in a SAS file in the coordinate list (COO) format. The label information is stored separately in SAS files in relational format.

Two types of base classifiers are used in the flat model. One is BOOLLEAR, which is implemented in the HPBOOLRULE procedure. The other is BOOLLEAR_H, a classifier that is derived from BOOLLEAR and

[5] http://lshtc.iit.demokritos.gr/files/second/datasets/wikipediaLarge2.0.tar.gz

that enables you to use hierarchy information to improve categorization accuracy when positive sample size is very small. The process for computing BOOLEAR_H is as follows:

1. Extract rules for the leaf node, its parent node, and its grandparent node on training data. If a node has multiple parents or grandparents, randomly choose one of each.
2. Build a meta-classifier based on the rule sets that are extracted for the leaf node, its parent node, and its grandparent node, as follows:
    a. Use the rule sets to predict on the validation data, which results in three sets of predictions.
    b. Construct meta-training data by taking the three sets of predictions as three meta-features and taking the true labels of the validation data as the targets. Use the LOGISTIC procedure to build a logistic regression model on the meta-training data. Save the model.
3. Use the rule sets that are extracted for the leaf node, its parent node, and its grandparent node to do prediction on the test data and generate three sets of predictions.
4. Apply the meta-classifier that was generated in step 2 on the three sets of predictions that were obtained in step 3 to generate the final prediction of each test document.
5. Output the final prediction.

When the size of the validation data is small, the logistic regression model that is built in step 2b might not be reliable. If you have more than one leaf node to model, you can use the following strategy to increase the size of the training data for building the meta-classifier:

1. Combine the meta-training data that are constructed for each leaf node. If you have *k* categories and there are *p* samples in each set of meta-training data, the combined meta-training data will have *k*p* samples. Each sample has three meta-features.
2. Use the LOGISTIC procedure to build a logistic regression model.
3. All the categories share the same meta-classifier for final scoring.

For each category, BOOLEAR_H needs to extract three rule sets instead of just one. Therefore, BOOLEAR_H is more computationally intensive than BOOLLEAR. However, BOOLEAR_H incorporates the information that is contained in the upper levels of the hierarchies, which might be very helpful when the number of positive documents is very small in the category.

PERFORMANCE OBSERVATIONS

BOOLLEAR and BOOLLEAR_H have been tested on the WIKI50 and WIKI500 problem sets. Linear support vector machine (SVM) is a powerful modeling tool for text categorization and is used as a baseline in the experiment.[6] The micro-average F1 and the macro-average F1 are used to measure the performance of algorithms. To compute the micro-average F1, your first compute the TP, FP, TN, and FN on each problem, average them, and then use the averaged TP, FP, TN, and FN to compute the F1 measure. To compute the macro-average F1, you first compute F1 for each problem and then average the individual F1 measures to obtain the macro-average F1 measure. The micro-average F1 and the macro-average F1 enable you to measure the performance of a classifier when it is applied to multiple problems. For BOOLLEAR, BOOLLEAR_H, and SVM, default parameters are used for model fitting. Table 1 shows the performance of the flat model when different base classifiers are used. The results reported in Table 1 are obtained on a Windows server that has two Intel Xeon L5530 CPUs and 64 GB of memory. In the experiment, the HPBOOLRULE procedure takes about 26 seconds on average to solve a problem.

The results presented in Table 2 show that on both WIKI50 and WIKI500, BOOLLEAR and BOOLLEAR_H obtained better performance than SVM. The 10 problems defined in WIKI50 and in WIKI500 are highly imbalanced, which makes them hard for SVM to handle. In contrast, BOOLLEAR and BOOLEAR_H are less sensitive to skewed data. BOOLLEAR_H achieves a better performance than BOOLLEAR on WIKI50, but not on WIKI500. This suggests that when the number of positive samples is very small, the information

[6] LIBLINEAR is used in the experiment. http://www.csie.ntu.edu.tw/~cjlin/liblinear/

obtained from upper levels of the hierarchy can help improve performance. But when the number of positive samples is relatively large, BOOLLEAR can obtain enough information and reliably model the category, so the information obtained from upper levels of the hierarchy becomes less useful. Experiment results show that BOOLLEAR is very effective for modeling imbalanced text data. And when positive sample size is small, incorporating information from the upper levels of the hierarchy can help improve performance.

Algorithms	WIKI50		WIKI500	
	micro F1	*macro F1*	*micro F1*	*macro F1*
BOOLLEAR	0.692308	0.639295	0.77193	0.76888
BOOLLEAR_H	0.719101	0.704642	0.77193	0.76888
Linear SVM	0.621622	0.544379	0.67470	0.67320

Table 2. Performance of the Flat Model When Different Base Classifiers Are Used

CONCLUSION

Hierarchical text categorization poses challenges when you categorize a large-scale text corpus that is organized under a complicated taxonomy. This paper introduces a novel technique, named BOOLLEAR, for text categorization. BOOLLEAR is implemented in the HPBOOLRULE procedure and in the SAS Text Miner Text Rule Builder node. BOOLLEAR is very efficient and performs robustly even when data are highly imbalanced and the number of positive samples is very small. BOOLLEAR can be used as a base classifier in the flat model and provides a powerful way for you to categorize large-scale hierarchical text corpora. Experiments fully demonstrate the effectiveness of BOOLLEAR for hierarchical text categorization.

REFERENCES

Cox, J., and Zheng, Z. 2014. System for efficiently generating k-maximally predictive association rules with a given consequent. US Patent 20140337271.

Manning, C. D., Raghavan, P., and Schütze, H. 2008. *Introduction to Information Retrieval.* Cambridge: Cambridge University Press.

Soka, R. R., and Rohlf, F. J. 1994. *Biometry: The Principles and Practice of Statistics in Biological Research.* New York: W. H. Freeman.

ACKNOWLEDGMENTS

The authors would like to thank Anne Baxter for her editorial contributions.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Zheng Zhao	James Cox	Russell Albright	Ning Jin
SAS Institute Inc.	SAS Institute Inc.	SAS Institute Inc.	SAS Institute Inc.
SAS Campus Drive	SAS Campus Drive	SAS Campus Drive	SAS Campus Drive
Cary, NC, 27513	Cary, NC, 27513.	Cary, NC, 27513	Cary, NC, 27513
Zheng.Zhao@sas.com	James.Cox@sas.com	Russell.Albright@sas.com	Ning.Jin@sas.com