Information Processing and Management 49 (2013) 484–496

Contents lists available at SciVerse ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Two scalable algorithms for associative text classification

Yongwook Yoon *, Gary G. Lee

Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), San 31, Hyoja-Dong, Pohang 790-784, Republic of Korea

ARTICLE INFO

Article history:
Received 23 September 2009
Received in revised form 27 September 2012
Accepted 30 September 2012
Available online 16 November 2012

Keywords:
Association rule mining
Associative classification
Text categorization
Large-scale dataset

ABSTRACT

Associative classification methods have been recently applied to various categorization tasks due to its simplicity and high accuracy. To improve the coverage for test documents and to raise classification accuracy, some associative classifiers generate a huge number of association rules during the mining step. We present two algorithms to increase the computational efficiency of associative classification: one to store rules very efficiently, and the other to increase the speed of rule matching, using all of the generated rules. Empirical results using three large-scale text collections demonstrate that the proposed algorithms increase the feasibility of applying associative classification to large-scale problems.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

Associative classification uses association rules which are mined from a transactional log database (Li, Han, & Pei, 2001; Liu, Hsu, & Ma, 1998). An association rule represents a co-occurrence relation among items in transactional logs. If an item in an association rule is a class label, the rule can be used for classification. Associative classifiers have been applied to various types of data, including text documents (Antonie & Zaïane, 2002; Li, Sugandh, Garcia, & Ram, 2007; Wang & Karypis, 2005; Yoon & Lee, 2008) and biological data such as DNA or protein sequences (She, Chen, Wang, & Ester, 2003). Whereas other types of classifiers including Naive Bayes and Support Vector Machine (SVM) consider only word features, associative classifiers can exploit *combined* features (for example, phrases) as well as words.

Words are elementary features in text classification. In a text collection, a document generally consists of hundreds of words. If a document is viewed as a sequence of word features, it exists in a very high-dimensional space. If phrase features are included as well as words, the dimensionality increases. A training database contains tens of thousands of documents, hence they become distributed sparsely in the document space. Therefore, raising the coverage for unseen test documents during the prediction phase is a big challenge. To match more test documents in associative classification, the length of word pattern in a rule needs to be reduced. Reducing the length of the pattern may degrade classification accuracy, although it can improve the coverage for test instances. Associative classifiers resolve this reduction in accuracy by choosing a smaller number of qualified rules and combining them in a predefined manner during the prediction phase (Antonie & Zaïane, 2002; Liu et al., 1998; Wang & Karypis, 2005). Promising results in text classification have been obtained by using a large number of low-order association rules, and applying a boosting technique to increase the classification accuracy (Yoon & Lee, 2008).

A large number of association rules can be mined by adjusting the mining parameters such as minimal support and minimal confidence to low values. However, too many association rules can hardly be processed in a real situation because they

* Corresponding author.
E-mail addresses: ywyoon@postech.ac.kr (Y. Yoon), gblee@postech.ac.kr (G.G. Lee).

0306-4573/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.ipm.2012.09.003

require very long computation time and very large storage space. Especially, generating high-order[1] rules exacerbates this problem because the number of rules produced grows exponentially with the order. We propose two new algorithms to process such a huge number of rules very efficiently. One algorithm represents association rules in a compact format so that it can save a large amount of storage space when a larger number of rules are mined to build classifiers. The other algorithm uses a new data structure to conduct the classifier building process very quickly.

When a large-scale text collection is processed, the number of generated rules may exceed 10^6, so they must be written into a disk file due to limitations in main memory. This paper proposes a novel method of representing class association rules in a compact manner where the itemset of a rule, the antecedent part of the rule, is represented in a compact format using the information of previously generated rules. Basically, the proposed classification method requires a large number of rules while some associative classification methods produce a small number of rules from the beginning. Our classification algorithm (Yoon & Lee, 2008) can achieve a maximum performance by generating as many rules as possible, which means minimizing the loss of the given information. The saving of 1 Gbyte in the file size cannot be negligible in real situations.

To apply the mined rules to a classification task, some qualified rules should be chosen from the original generated rules. This process is called *classifier building* or *rule pruning* (Antonie & Zaïane, 2002; Liu et al., 1998; Wang & Karypis, 2005). The compact rule representation also help to perform this classifier building process efficiently. When the building process starts, the rules are loaded into memory again and are sorted in the order of confidence and support. They keep their compact form because only confidence and support information are needed in the sorting process (uncompressed antecedent information is required in the rule-matching process). Additionally, in-memory sorting requires an extra amount of memory. As the compact rules themselves occupy less memory space, the sorting or classifier building process can be performed more efficiently.

The second algorithm allows the process of rule matching against training documents to be performed quickly. To store training documents and to perform rule matching, we propose a new data structure which is specially designed for our classifier building process. One strong point of the second algorithm is the ability to delete documents from the structure efficiently as well as the matching speed. Using a naive matching algorithm takes $O(klNM)$ time, where k is the average length of a rule, M is the number of the rules, l is the average length of a test document, and N the number of the training documents. Our proposing algorithm can finish the process in one order of magnitude less time compared with the original method. Empirical results using large-scale document sets demonstrate that the proposed algorithms make associative text classification scalable to real-world problems.

This paper is structured as follows. Section 2 lists the previous studies which handled the efficiency issues related to the associative classification. Section 3 introduces associative text classification and formulates the problem. Section 4 describes a new model for representing association rules compactly, and explains the algorithm for efficiently matching rules to the training documents. Section 5 shows the experimental results using the proposed algorithms and compares them with the results obtained using previous methods. Finally, Section 6 concludes the paper.

2. Related works

Liu et al. (1998) introduced CBA, a prototype of associative classification. It adopts a two-stage induction process of classification rules. First, it generates Class Association Rules by *Apriori* method. Then, it applies each of rules to the set of training examples. The rules which classify correctly at least one example are selected and the covered examples are deleted from the database. While CBA deletes the examples when they are covered once, CMAR (Li et al., 2001) postpones the deletion until the examples are covered m times, which can improve the coverage of the selected rules. However, the postponement of the deletion means that the algorithm performs additional $m - 1$ times of rule matching against the example. In the associative classifier building process (Yoon & Lee, 2008) where our rule matching algorithm is used, there may be far more than m times of matching per training example, which requires a more efficient matching algorithm for an enormous number of association rules.

HARMONY (Wang & Karypis, 2005) keeps a rule with the highest confidence, which is assigned to each covering training instance. Without an additional pruning procedure, it can generate a small number of high-confidence rules very efficiently. However, such a one-rule-per-example principle may work adversely when handling a database with uneven distribution of class labels, because a class with many training examples may have a high prediction score on that class label, which leads to a wrong prediction.

Frequent closed itemset model (Pasquier, Bastide, Taouil, & Lakhal, 1999) was used for a compact representation for association rules. Without explicitly producing the subset rules, we can derive entire rule set from the closed form at a later time. Therefore, the rule mining process adopting the frequent closed itemset model can be finished within a less time and space than other mining methods. However, as far as we consider the classification task with the rules, it is a different story. In our classification method, we use all subset rules of a frequent closed rule because it is far more possible to match test documents if a rule has a smaller number of items. Normally, a rule in a closed itemset format is long and compact, and cannot be used in its own in our classification method; they should be expanded into its subset rules. In our rule mining method, a rule has a compact antecedent part and does not include any other subset rules in its compact form. From the beginning, we did not consider combining several rules into one because we mine rules for the purpose of classification.

[1] The order of a rule denotes the length of the word pattern.

In the area of associative classification, there have not been many studies on the rule matching algorithm itself. In the document retrieval domain, an inverted index (Zobel, Moffat, & Ramamohanarao, 1998) is used to structure documents and match query phrases against the documents in the database. Although it provides a fast rule matching, it suffers from severe performance degradation when it deletes documents and restructures the index. We made a performance comparison between the inverted index and our method in the Experiment section. The suffix trees (Weiner, 1973) lead to a linear time algorithm in query matching and index construction (Sandhya, Lalitha, Govardhan, & Anuradha, 2011). If we are to use the suffix tree to represent training documents, we need to sort the word items of each document according to their frequencies in the document set, so that the order of items in a rule can be kept the same as in the documents. Moreover, the suffix tree based rule matching would not be more efficient because the rule does not need to be a *substring* of the document in order to get a match.

3. Associative text classification

3.1. Overall flow

Most associative classifiers follow a similar overall system architecture for associative text classification (Fig. 1). They begin by processing raw text data for rule mining. Each document is converted to a transactional record format (step 1), and this preprocessed database is mined for frequent patterns, i.e., class association rules (step 2). From that large number of generated rules, a smaller number of qualified rules are selected and combined into a final rule set for classification (step 3). When a new document is to be classified, it is converted to a pattern of words and matched to the classification rules (step 4). According to the matching score, classes are assigned to the test document (step 5).

3.2. Association rule mining

Association rules were initially devised to represent a degree of *association* between items in a transactional log database (Agrawal & Srikant, 1994). A document corresponds to a transactional log record, and the words in it to the items of the transaction. Let d be the average length of training documents, and V be the vocabulary of the document set. Then, $X = X_1 \times X_2 \times \cdots X_i \cdots \times X_d$ denotes a document, where $X_i \in V$. Let $Y = \{c_1, c_2, \ldots, c_{|Y|}\}$ be the set of class labels. The set of training examples D can be written as

$$D = \{(x, y) | x \in X,\; y \in Y\}.$$

A class association rule (CAR) is a mapping from a set of items to a class label. Let Z denote an itemset (or word pattern), $Z = Z_1 \times \cdots Z_i \cdots \times Z_k$, where $Z_i \in V$. Then, a CAR can be written as

$$Z \rightarrow Y.$$

k is called the *order* of the rule, and generally $k \ll d$. The left-hand side of the arrow is called the antecedent, and the right-hand side the consequent.

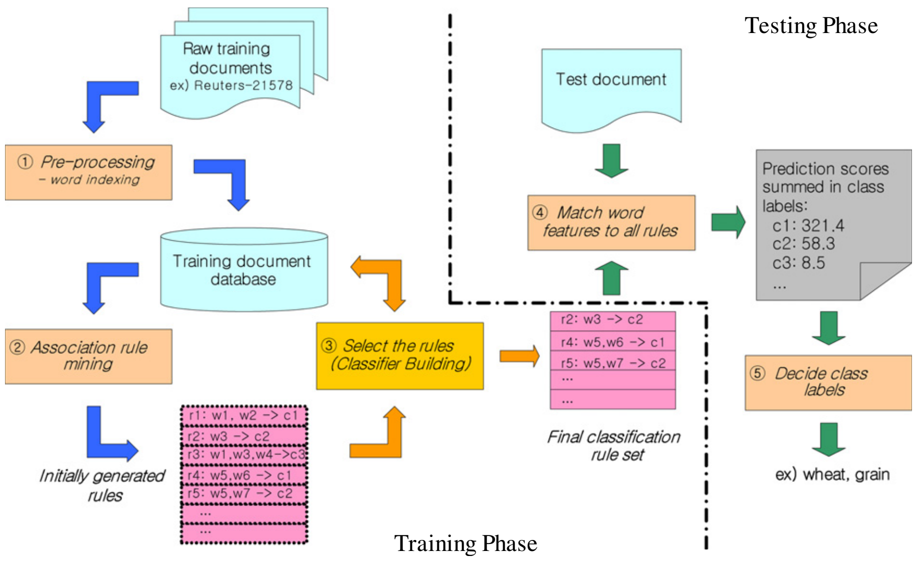

Testing Phase
Raw training documents ex) Reuters-21578
① Pre-processing – word indexing
Training document database
② Association rule mining
Initially generated rules
r1: w1, w2 -> c1
r2: w3 -> c2
r3: w1,w3,w4->c3
r4: w5,w6 -> c1
r5: w5,w7 -> c2
...
...
③ Select the rules (Classifier Building)
r2: w3 -> c2
r4: w5,w6 -> c1
r5: w5,w7 -> c2
...
...
Final classification rule set
Test document
④ Match word features to all rules
Prediction scores summed in class labels:
c1: 321.4
c2: 58.3
c3: 8.5
...
⑤ Decide class labels
ex) wheat, grain
Training Phase

Fig. 1. Text classification using associative classification rules; left: training phase; right: testing phase.

Definition 1. The *support* of a CAR is the number of training examples in which the pattern and the class label of the rule occur.

Definition 2. The *confidence* of a CAR is is the support of the CAR divided by the support of its antecedent.

When mining association rules, the mining algorithm checks the support and the confidence of generated rules, and filters out the rules whose support and confidence are less than certain threshold values, *minimum support* (*min_sup*) and *minimum confidence* (*min_conf*). The following is an example of class association rules mined from a document collection:

$$\text{rate},\ \text{bank} \rightarrow \text{interest}\ (151,\ 0.31), \tag{1}$$

where 151 is the support and 0.31 is the confidence of the rule. This rule indicates that the training documents which contain both words "rate" and "bank" occur 151 times, and that 31% of them include the category "interest".

3.3. Classifier building

Mining frequent patterns generates numerous CARs. Those rules are not used directly in the prediction phase because the number of rules is too large and the prediction accuracy is not good. Instead, a target classifier is built by choosing a smaller number of qualified rules, which are identified by validating the initially-generated rules against training documents (Antonie & Zaïane, 2002; Cheng, Yan, Han, & Hsu, 2007; Li et al., 2001; Liu et al., 1998; Wang & Karypis, 2005; Yoon & Lee, 2008).

In a typical algorithm for classifier building (Fig. 2), the generated rules are first sorted in a predefined order. Then, starting from the highest-ranked rule, they are matched to training documents rule-by-rule. The rule-matching operations (Fig. 2, nested For loop) consume most of the computation time. The rule that classifies a training document correctly is chosen as a member of a final set of classification rules (line 3-b-i), and training documents classified by the rule are deleted from the database (line 3-b-ii).

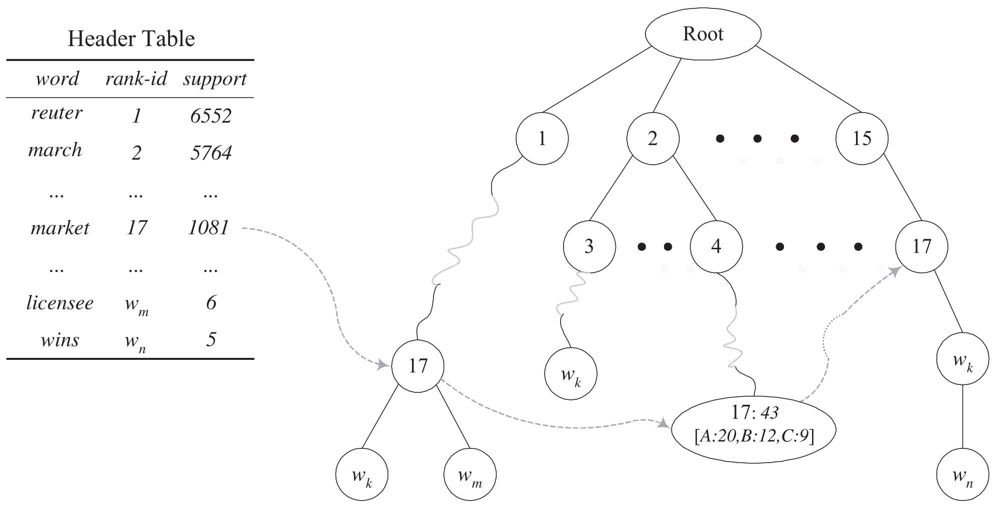
4. Two efficient algorithms

4.1. Memory-saving representation of rules

There are different methods to generate association rules from a database. Our approach is based on the *FP-growth* method (Han, Pei, & Yin, 2000). First, the algorithm scans the database and constructs a prefix tree that represents all training documents. The items of the training documents are encoded as integers and stored in a node of the FP-tree (Fig. 3). The item

```
Algorithm ClassifierBuilding
Input Training database: $D_0 = \{(x_i, y_i)\}_{i=1}^{N}$,
      Class Association Rules: $R = \{r_1, r_2, ..., r_M\}$,
Output Final rule set: $H$
Begin
  1. sort $R$ to the descending order of confidence and support
  2. $H \leftarrow \{\}$
  3. For $t = 1, 2, ..., M$ //rule index
       For $i = 1, 2, ..., |D_{t-1}|$ //document index
       (a) Apply $r_t : z_t \rightarrow y_t$ to document $(x_i, y_i)$.
       (b) If $r_t$ classifies correctly, then
           i. Select $r_t$ as a member of the final rule set $H$:
              $H \leftarrow H \cup \{r_t\}$
           ii. Delete $(x_i, y_i)$ from $D_t$
              $D_t \leftarrow D_{t-1} - \{(x_i, y_i)\}$
End
```

Fig. 2. Algorithm for building a final classifier.



Header Table

word	*rank-id*	*support*
reuter	*1*	*6552*
march	*2*	*5764*
...	...	...
market	*17*	*1081*
...	...	...
licensee	w_m	*6*
wins	w_n	*5*

Fig. 3. FP-tree and a header table after scanning training documents. A number inside a circle denotes a word-rank id.

with the largest support value is encoded as ‘1’. A path from the root node to a terminal node denotes a training document, and the nodes in the path are linked in ascending order of their integer codes.

Each node also contains additional information including the support of a word item and the supports of classes assigned. The oval node in the bottom of Fig. 3 shows such information: word pattern $\{2, 4, \ldots, 17\}$ occurs 43 times in the training documents; of these instances, class *A* occurs 20 times, class *B* occurs 12 times, and class *C* occurs nine times.

After constructing a global FP-tree for the training documents, the algorithm generates association rules while recursively constructing sub FP-trees which are conditioned on a frequent word pattern that was mined previously. In this method, the number of generated rules increases exponentially with the depth of the recursive calls. Thus, the number of generated rules plus local variables accumulated by repetitive recursive calls can become too large to be stored in main memory. The generated rules are written into a disk file, but occasionally even that file can exceed the disk's capacity.

In the FP-growth method, frequent patterns are output redundantly as they have identical conditioning suffix of frequent patterns mined; this redundancy contributes significantly to the memory requirement. This paper proposes a rule-encoding and -decoding algorithm that reduces this redundancy and allows a huge number of association rules to be stored very efficiently. Our scheme represents a class association rule as follows:

$$w_1\ w_2\ \ldots\ w_k > supp_a\ c_1\ s_1\ c_2\ s_2\ \ldots\ c_m\ s_m, \tag{2}$$

where w_i denotes the words in the rule, $supp_a$ the support of the antecedent, c_j is the class label, and s_j is the support of the rule. In effect, Eq. (2) represents *m* rules that have identical antecedent words. The confidence of the rule with class label c_j is $s_j/supp_a$. The *k*-order word pattern can be further compacted using our first proposed algorithm (Fig. 4).

Procedure SuffixOmission() saves only non-repeating parts of antecedent words *c*[ ], instead the whole itemset *w*[ ]. To do this, the procedure keeps the previously-input words in buffer *b*[ ] whose size is fixed to the maximum allowed rule length *K*. Before generating association rules, all elements of *b*[ ] are initialized to zero. In SuffixOmission(), before the elements of input words *w*[ ] are compared with the buffer elements *b*[ ] (line 1), they are sorted in ascending order of the item codes of the header table (Fig. 3).

Encoding results for a bit of association rules were generated from Reuters-21578 document collection (Apté, Damerau, & Weiss, 1994) (Table 1). Rule-order parameter *K* was set to 4 in this case. Each input of antecedent words (left column) is generated sequentially by the FP-growth algorithm, and the right column is output by SuffixOmission(). Most antecedent patterns can be encoded as just one item, rather than as *k* items (except for the case “2 5 17”), and this reduction in items saves a great amount of storage. In the expression (2), class labels c_j are also coded as integers starting from zero: thus an exemplary compact form would look like:

3 > 288 0 144 3 94,

which corresponds effectively to two class association rules. When decoding a compact representation (Eq. (2)) into a normal representation (Eq. (1)), procedure SuffixRestoration() is called, which is simply SuffixOmission() run in reverse.

Theorem 1. *The compact representation that is output by procedure SuffixOmission() contains all information about the association rules generated by the FP-growth method.*

```
Procedure SuffixOmission(w,b,c)
Input itemset to be output: $w[\,] = \{w_1, w_2, \ldots, w_k\}$,
    word itemset buffer $b[\,]$ of fixed size $K$
Output compact word itemset $c[\,]$
Begin
  1. sort items in $w[\,]$ to the order of the initial header table
  2. For $i = k, k-1, ..., 1$ //from the biggest item
     (a) $j = K - (k - i)$ //from the biggest item
     (b) If $w[i] \neq b[j]$, then
           i. copy $w[i..1]$ to $b[j..(j - i + 1)]$
          ii. fill 0 into $b[(j - i)..1]$
         iii. break the For loop
  3. copy $w[1..i]$ to $c[\,]$ //only output non common prefix
End
Procedure SuffixRestoration(c,b,w)
// arguments $c[1..t]$ and $b[\,]$ are for input, and $w[\,]$ for output
Begin
  1. For $j = K, K-1, ..., 1$ //from the biggest item in the buffer
     (a) If $b[j] < c[t]$,
         then break the For loop //determine the element to be replaced
  2. copy $c[]$ to $b[(j - t + 1)..j]$ //replace buffer content
  3. If $(j - t + 1) > 1$
     then fill $b[1..(j - t)]$ as zeros //fill leading zeros
  4. copy $b[(j - t + 1)..K]$ to $w[\,]$ //$w$ is recovered to a non compact form
End
```

Fig. 4. Algorithm to decode and encode antecedent words efficiently.

Table 1
Encoding antecedent words into a compact form.

Input	Buffer	Output
...	[12 14 15 16]	...
[13 14 15 16]	[13 14 15 16]	[13]
[17]	[0 0 0 17]	[17]
[1 17]	[0 0 1 17]	[1]
[4 17]	[0 0 4 17]	[4]
[1 4 17]	[0 1 4 17]	[1]
[2 4 17]	[0 2 4 17]	[2]
[1 2 4 17]	[1 2 4 17]	[1]
[3 4 17]	[0 3 4 17]	[3]
[1 3 4 17]	[1 3 4 17]	[1]
[2 3 4 17]	[2 3 4 17]	[2]
[2 5 17]	[0 2 5 17]	[2 5]
[3 5 17]	[0 3 5 17]	[3]
...	...	...

Proof. Because FP-growth processes items in descending order of their support values, and because procedure SuffixOmission() stores the compactly-encoded rules *c*[ ] line-by-line in a file, SuffixOmission() can conserve the order in which the word patterns of the rules are generated. When an item is conditioned to generate a composite itemset in the FP-growth algorithm, FP-growth always inspects item nodes in a bottom-up manner: the items which will be inspected always have higher ranks (smaller integer codes) than the conditioned item. Procedure SuffixRestoration() searches the internal buffer

$b[\ ]$ for a smaller item code than the current compact code $c[i]$ (line 1), and replaces it with $c[\ ]$ if it finds a smaller code, otherwise SuffixRestoration inserts $c[\ ]$ at the front (line 2) so that the words in $b[\ ]$ can be kept in the generation order of FP-growth. Therefore, the conditioned item always exists in the buffer, and the whole buffer content coincides with the output originally generated by FP-growth. □

4.2. Fast rule matching mechanism

The process of matching the rules to training documents takes a considerable time (Section 3.3). This time can be greatly reduced by adopting a sophisticated data structure and an algorithm that applies it. A set of training documents is rearranged into a graph structure in which the word items are indexed and linked to each other (Fig. 5). First, the word items of each training document are coded using their rank-id in the header table of the initial FP-tree (Fig. 3). Then, they are sorted in ascending order of the codes, and stored into a list structure. Root node *TrainDoc* has a link to the word list (or a training document). A word node has a word rank code and a class label of the document. After all the training documents are converted to a set of lists, each item in each list is indexed using a global index table, by linking an item entry from the table to the item nodes of the document lists one-by-one. This linking process resembles that of the FP-tree construction.

To determine whether a rule can classify a document correctly, the rule must be verified to be a subset of the document. If every element of the words in a rule occurs in the document, then the rule is a subset of the document. Matching a rule with k words to a document with l words using a naive method takes $O(kl)$ time in the worst case, and the total time to process M rules with respect to N training documents reaches $O(kM \cdot lN)$. We propose an algorithm (Fig. 6) which allows N to be over two orders of magnitude smaller than it is in the naive algorithm, using the proposed data structure (Fig. 5). In addition, l can also be greatly reduced.

Algorithm **ReverseRuleMatching** matches a rule to N training documents. The words in the rule are sorted, then the last word of the rule is compared to the last word of a document, and the comparison proceeds to the words with higher ranks. The variable q (line 2) is a pointer that indicates the word node of the next document to which rule r will be compared. The *While* loop of line three iterates with respect to the documents which are linked by pointers, starting from the entry in W. For example, in Fig. 5, documents which have words labeled as '349' appear in the first, third, and final locations in the training set, and are examined in turn. The *While* loop (line 3-e) determines whether the word list of r is a subset of the current document $v[\ ]$ (word indexes i and j point to upper nodes in turn, along the path of the lists). If this checking is finished in the middle of the word lists, the algorithm stops the loop prematurely (the *break* statements), which reduces the number of comparisons greatly.

If training documents being compared are judged to support r, they are included in set *ClsTrDoc* (line 3-f). In Fig. 5, the first and the last document would be classified correctly by r. If rule r classifies at least one training document correctly, r is included in the final classification rule set (line 4). Then, the documents in *ClsTrDoc* are examined to be deleted from the database (line 5). In the simplest case, a document which is hit once by a rule is pruned (Fig. 2, line 3-b-ii). When this pruning occurs, the number of remaining documents diminishes very quickly. Several different methods give documents a second

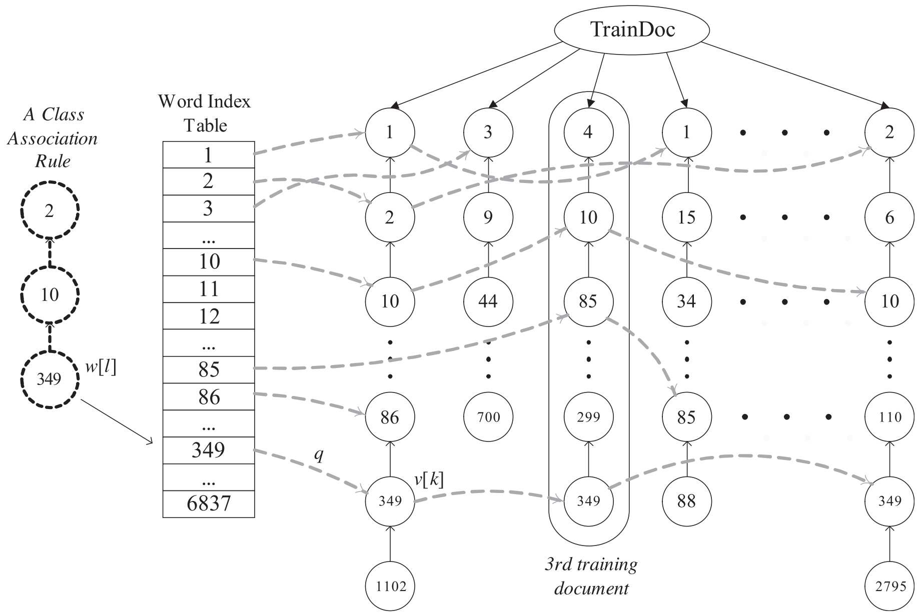


Fig. 5. Training documents indexed by the word items.

```
Algorithm ReverseRuleMatching
Input Training database $D = \{(x_i, y_i)\}_{i=1}^{N}$, organized as a indexed greaph;
      Class Association Rule $r = \{w_1, w_2, ..., w_l, c_r\}$, words and class label
      (the words in the rule are sorted to the order of high supports);
      Word index table $W$; Hit score threshold $\theta$
Output membership of $r$ in the final classification rule set
Begin
  1. $ClsTrDoc \leftarrow \phi$ //set of correctly classified training documents
  2. $q \leftarrow$ the entry of $W$ indexed by the last word $w[l]$ of $r$
  3. While $q$ is not null do
     (a) $subset \leftarrow FALSE$
     (b) $k \leftarrow$ the index of the word node directed by $q$
     (c) $i \leftarrow l$ // $i$: index of rule words
     (d) $j \leftarrow k$ // $j$: index of document words
     (e) If $c_v = c_r$, then //$c_v$: the class label of the document
         While not the end of $w[\ ]$ do // $w[\ ]$: rule word list
           i. If $w[i] = v[j]$, then // $v[\ ]$: document word list
                • $i \leftarrow i - 1, j \leftarrow j - 1$ //move to the upper node
                • If $i = 0$ //the last node?
                  then $subset \leftarrow TRUE$ and break
              else if $w[i] > v[j]$ //$w[\ ]$ is not subset
                   then break
              else //$w[i] < v[j]$
                   $j \leftarrow j - 1$ //move to the upper node
     (f) If $subset$
         then $ClsTrDoc \leftarrow ClsTrdoc \bigcup \{v[j].did\}$ //$did$: document id
     (g) $q \leftarrow v[k].next$ // link to the next document
  4. If $ClsTrDoc$ is not empty
     then include $r$ into the set of classifying rules
  5. For each document $d$ in $ClsTrDoc$ do
     (a) $d.score \leftarrow d.score + r.conf$
     (b) If $d.score > \theta$
         then delete $d$ from $TrainDoc$
End
```

Fig. 6. Algorithm to match the rules to training documents in a fast manner.

chance of surviving the pruning process although they have been classified correctly. One accumulates the confidence score of the rules which classified the document correctly, then compares the accumulated score with a given threshold value (θ) to determine whether the document should be deleted (Yoon & Lee, 2008).

Increasing the hit score threshold θ reduces the rate at which documents are deleted. Although this process requires more time in the classifier building process, classification accuracy can be better than that achieved using the simple deletion method (Antonie & Zaïane, 2002; Li et al., 2001; Yoon & Lee, 2008). The optimal value for the score sum threshold θ is determined by a sort of grid search for some interval using an extra training data. The graph structure of the training documents (Fig. 5) could be formed as a prefix tree like an FP-tree, instead of the proposed redundant list structure. Although the prefix tree has the merit of reducing the amount of memory required for storage of words, it shows poor efficiency in deleting training documents and reorganizing node links.

We tried applying another data structure to conduct rule matching fast. The training documents are stored with an inverted index. Then, as in the document retrieval task, given the words of a rule, matching documents are returned. The rule matching algorithm in the classifier build process using this inverted index approach is very simple (Fig. 7). The document matching is conducted efficiently because it uses a word index similar to q in Fig. 5. In addition to the inverted index, a variable (*d.score* in line 4(a) of Fig. 7) is used for recording the hit score of a training document. The line 4(a) updates the score of

```
Algorithm InvertedIndexMatching
Input $D$(with the inverted index), $r, W, \theta$
Output membership of $r$ in the final classification rule set
Begin

  1. $h[\,] \leftarrow$ the list of documents matcing $w[l]$ and $c_r$
     // $h[\,]$ is the documents returned by the inverted index matching
  2. If $h[\,]$ is null
     then Stop the algorithm
  3. $classify \leftarrow FALSE$
  4. For each document $d$ in $h[\,]$ do
     (a) $d.score \leftarrow d.score + r.conf$      // updates the score
     (b) If $d.score > \theta$      // compare with the threshold
         then continue      // a deleted document!
     (c) $classify \leftarrow TRUE$
  5. If $classify$
     then include $r$ into the set of classifying rules

End
```

Fig. 7. Algorithm to match the rules to training documents using the inverted index.

a document each time it is covered by a rule. In line 4(b), when the score exceeds the threshold, the document is to be deleted. This algorithm does not delete the document but continues the processing for the next covered document. We could have deleted the document from the training set, and then reconstruct the inverted index structure of the document set. However, this frequent re-indexing would cause an enormous increase in computing time. We present the experimental results of rule matching using the inverted index in the next section.

Before showing the experimental result, we explain the effect of the distribution of words in a document collection. Fig. 8 shows the distribution of the support of the words in Reuters-21578, which has 7193 training documents. The *x*-axis represent the 6837 words in the training set, ordered by their support ranks. Most of the words have a support value under 100. In other words, a word which appears in a rule has a very low probability of occurring in the training documents. The pie chart in Fig. 8 represent this well: words which occur in a training document with probability $p(w)$ larger than 0.01 have only 13% share in the entire vocabulary of the training documents. Our rule matching algorithm makes a good use of this distribution. In each iteration of matching to the document set, only about a thousandth of the entire document set is considered. In addition, the volume of the document set continues to decrease as documents being hit many times are

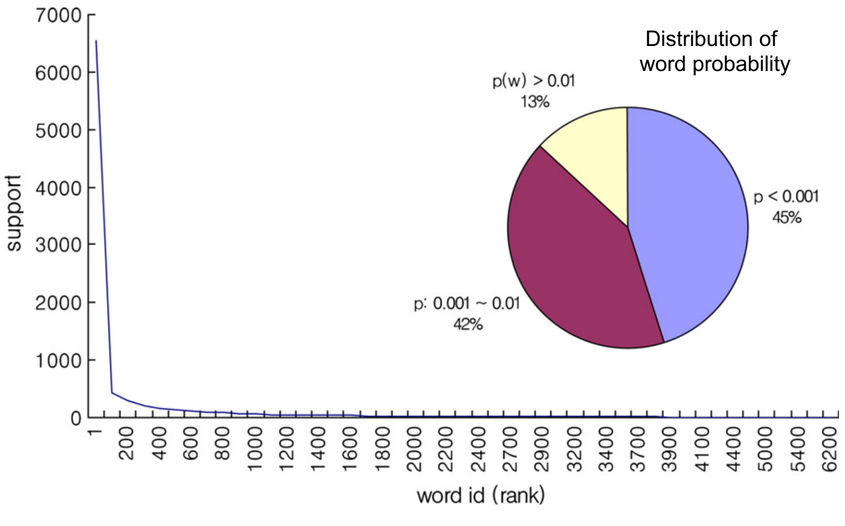


Fig. 8. Distribution of the word support in the training documents.

Table 2
Statistics on the text databases used in the experiment.

	Reuters-21578	20 Newsgroups	Chef Moz
#documents	9979	19,997	88,171
#train: #test	7193: 2786	14,997: 5000	88,171: 0
vocabulary size	22,424	90,833	101,840
#words/doc	49	77	37
#classes	10	20	16

deleted from the set. Those can decrease the time complexity of rule matching, $O(kM \cdot lN)$, greatly by reducing N (the number of the training documents) to about one thousandth in average.

5. Experiment

5.1. Dataset

Three multi-class test collections were used for large-scale text categorization. *Reuters-21578* is a collection of articles from the Reuters newswire; the *ModApte* split version (Apté et al., 1994) was used, specifically, documents in the 10 TOPICS categories. The *20 Newsgroups* collection (Lang, 1995) is a collection of USENET mail postings from 20 discussion groups. A much larger dataset, the Chef Moz collection from ODP[2] was used. From this dataset, we selected documents of the top 16 categories based on the style of cooking. Statistics on the three collections (Table 2) indicate that they are large enough to test the scalability of our two algorithms.

We used the BOW-toolkit (McCallum, 1996) to preprocess the documents. In a document text, the header part except for the title was removed. We filtered out general stop words and conducted no stemming. Our associative classification system was implemented with C++ and Perl codes executed on a Linux machine with 4 GB memory and 2.8 GHz CPU speed. Source codes of BCAR can be downloaded at http://isoft.postech.ac.kr/~ywyoon/BCAR.

5.2. Parameter selection

We conducted a simulation to investigate the relation between the number of rules and the classification accuracy in the associative classification, using the Reuters-21578 collection and setting the maximum order of generated rules to three (Fig. 9). BCAR (Yoon & Lee, 2008) was used as classifier, and the accuracy was measured using Breakeven Point (BEP).

The number of generated rules increased as *min_sup* and *min_conf* decreased (Fig. 9a). Because our purpose is to increase the classification accuracy by assuring that as many rules as possible are matched to a test document, the minimum support and the minimum confidence thresholds were set to low values. The *min_sup* and *min_conf* values in this experiment (Table 3) are not necessarily equal to the values which leads to the best classification accuracy, but were chosen just as to express well the effect of the storage saving.

Although using a large number of classification rules produced the best classification performance (Fig. 9b), it also required considerable computing time and storage capacity; this disadvantage should be resolved. Increasing the value of the maximum allowed rule order is not feasible due to generation of an excessive number of rules. In this experiment, rule order is set to $\leqslant 4$.

5.3. Results

5.3.1. Compact representation of rules

In order to measure the effect of representing rules in a compact form (Section 4.1), the sizes of rule files were compared when applying two representation methods (Table 3). In the *Naive coding* method, the antecedent part contains the original length of words and the consequent part the class label. In the *Suffix Omission* algorithm (Fig. 4), rules are represented in an abbreviated form as in the expression (2). In the table, we presented the number of rules, and compared the size of the files for the two methods, varying rule orders from one to four.

The Suffix Omission method produced rule files smaller than those of Naive coding by a great amount. The compaction ratios increased with the order of rules (Table 3). Even when the rule order was one, the file size produced by Suffix Omission was smaller than produced by Naive coding, because Suffix Omission writes several class labels in one line. When rule order was four, the file size produced by Suffix Omission was about half of that produced by Naive coding. This result implies that more classification rules can be generated in the same storage space using Suffix Omission than using Naive coding. On the total processing time which is taken for encoding/decoding and file writing, there is little difference between the Naive coding and the Suffix Omission algorithm.

[2] http://www.dmoz.org.

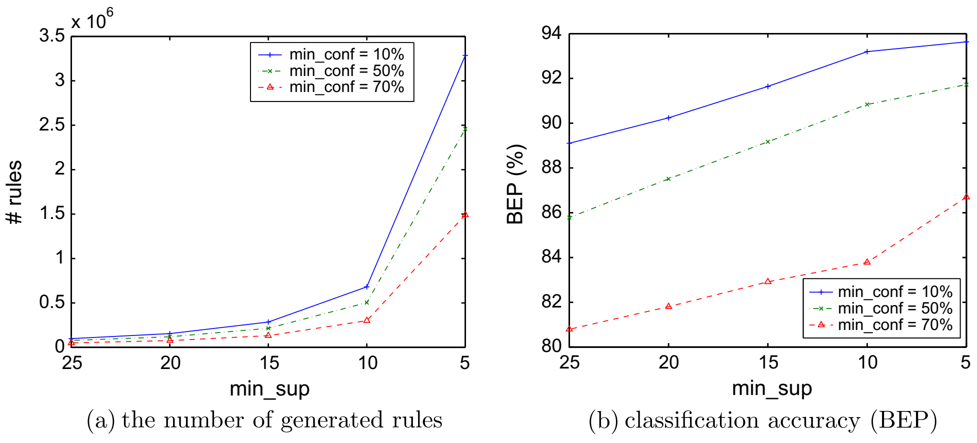


Fig. 9. Relation between the number of rules and the classification accuracy.

Table 3
Effect of compact representation for rules. *r_order* denotes rule order.

Data	r_order	#rules	File size (KB)		Processing time (s)	
			Naive coding	Suffix Omission	Naive coding	Suffix Omission
Reuters-21578 (min_sup = 5, min_conf = 0.2)	1	5846	84	68	41	41
	2	303,070	5101	3267	268	274
	3	3,891,581	75,050	40,926	1002	1020
	4	25,549,279	558,245	266,129	2950	3009
20 Newsgroups (min_sup = 10, min_conf = 0.05)	1	22,156	355	229	329	341
	2	504,839	9389	5712	691	702
	3	3,765,663	81,315	46,962	1303	1350
	4	21,412,862	550,563	285,298	3103	3168
Chef Moz (min_sup = 5, min_conf = 0.10)	1	25,126	380	267	411	415
	2	2,102,122	37,567	21,545	912	940
	3	91,746,147	1,899,689	958,581	8432	8506

We conducted another experiment to show the efficiency of our Suffix Omission algorithm, compared with another rule compaction method, the frequent closed itemset (FCI) model. In this experiment, we used the test subset of the Reuters-21578 collection, set *min_conf* to 50%, and put no limit on the maximum rule order. Varying the *min_sup* value, we examined the files size and the processing time for the outputs of the two methods (Table 4). Note that the FCI tool package (Zaki, 2004) which we used in the experiment does not include a subset-expansion code, but we implemented that part at the end of the original code for our experiment. Thus, the figures in the column under Frequent Closed contain additionally the values when a closed form of rules are expanded to all their subsets. The file size of the FCI model is five to ten times larger than that of the Suffix Omission method. In the case of the processing time, until we have *min_sup* 8 starting from 12, there is three times difference at most. However, if we handle a very large volume of rules such as in the case of *min_sup* 5, the difference ratio exceeds ten times. The Suffix Omission can process a large scale data more efficiently than other methods.

5.3.2. Time reduction in classifier building

Computing times for building classifier using two methods, the Naive matching and the ReverseRuleMatching algorithm, were compared (Table 5). Test data sets and the rule generation parameters were the same as in Table 3. Another parameter was added to the classifier building process: a threshold for document hit score (θ). Four values (1, 10, 50, and 100) in θ were combined with each rule order.

The computing time in Table 5 represents the total time for the classifier building process, including the time for reading rules and training documents and for sorting the rules. The rule matching procedure consumes most of the computing time. The total rule matching time increases with the number of rules to be matched, and with θ. The amount of time reduction by ReverseRuleMatching is remarkable: the ratio of time required by the Naive algorithm to that required by ReverseRuleMatching (reduction ratio) ranges from 3 to nearly 274.

The reduction ratio depends on the type of text collection and the classifier learning parameters (*min_sup*, *min_conf*, and θ). On the whole, ReverseRuleMatching reduced computation time more for 20 Newsgroups than for Reuters-21578. The vocabulary size of 20 Newsgroups is four times large as that of Reuters, whereas the training set is only twice as large (Table 2). Thus, the words of 20 Newsgroups are more-sparsely distributed in the training database than those of Reuters. Because

Table 4
Comparison of compact representations: Suffix Omission vs. frequent closed itemset.

Data	min_sup	#rules ($\times 10^3$)	File size (MB)		Processing time (min)	
			Suffix Omission	Frequent Closed	Suffix Omission	Frequent Closed
Reuters-21578 (#records = 2545, min_conf = 0.5, r_order = *unlimited*)	12	1917	24	193	<1	3
	10	3452	43	379	<2	6
	8	8531	103	1013	4	12
	5	614,564	10,661	60,897	197	2220

Table 5
Comparison of computing times in the classifier building.

Data	r_order	#rules	θ	Time (s)		
				Naive matching	Reverse matching	Ratio
Reuters-21578	2	303,700	1	21	7	3.0
			10	40	10	4.0
			50	83	22	3.8
			100	133	39	2.7
	3	3,891,581	1	332	72	4.6
			10	487	83	5.9
			50	928	114	8.1
			100	1199	147	8.2
	4	25,549,279	1	1378	602	2.3
			10	1756	635	2.8
			50	2313	729	3.2
			100	2748	812	3.4
20 Newsgroups	2	504,839	1	108	20	5.4
			10	261	33	7.9
			50	720	100	7.2
			100	1417	172	8.2
	3	3,765,663	1	290	84	3.5
			10	1652	109	15.2
			50	3196	219	14.6
			100	8168	358	22.8
	4	21,412,862	1	3299	729	4.5
			10	6421	799	8.0
			50	12,335	1002	12.3
			100	14,975	1202	12.5
Chef Moz	2	2,102,122	1	7697	30	257
			10	12,872	47	274
			50	25,202	128	197
			100	34,211	202	169
	3	91,746,147	1	235,975	293	805
			10	296,911	376	790
			50	373,926	609	614
			100	493,102	911	482

of this difference, 20 Newsgroups has a smaller number of documents being linked for rule matching compared to Reuters-21578, which results in more time reduction in the classifier building process.

Time reduction is most remarkable in the Chef Moz dataset. The vocabulary size of Chef Moz is about the same as that of 20 Newsgroups, but the number of training documents is six times larger than that of 20 Newsgroups. Hence, the number of rules is four times bigger than 20 Newsgroups in the case of 2nd order rules. (We did not list the result of 4th order rules because the number of the rules was so huge that they could not be generated within a reasonable time.) The time of the Naive matching in the Chef Moz data is very long, but the time of the Reverse rule matching was very short. The time reduction ratio even reaches several hundreds. This shows that the larger the dataset is the more positive effect our Reverse rule matching algorithm can acquire.

The experimental result using the document set structured by the inverted index was presented in Table 6. The *min_sup* and *min_conf* was 3 and 0.06 respectively. The computing times of the inverted index method did not change as θ increased, whereas the time of the Reverse rule matching increased linearly. The efficiency performance of the inverted index method was a little better than that of the Naive method due to its word index structure. However, it was still inferior compared to

Table 6
Comparison of computing times in the classifier building.

Data	r_order	#rules	θ	Time (s)		
				Inverted index	Reverse matching	Ratio
Chef Moz	2	4,471,772	25	27,017	87	311
			50	26,991	144	187
			100	24,797	237	105
			200	24,780	261	95
			400	24,772	596	42

that of the Reverse rule matching. The reason is that the size of the training document set, N, does not decrease as the rules are matched, although the document retrieval can be processed fast; matching documents just have their scores increased, but are not deleted from the inverted-index-structured document set.

6. Conclusion

Associative classification using a large number of classification rules can show good performance when applied to a large-scale database such as a text collection. However, such a large set of generated rules can cause computational efficiency problems, including shortage of memory space for storage of rules and an excessive amount of time to process rules and documents. We propose two new efficient algorithms to address those efficiency-related problems. By storing rules using a very compact representation format, a considerable amount of storage space can be saved when a huge number of association rules are processed for classifier building. By constructing a training document set as an indexed document list structure, and by matching rules to the documents indexed by the word with the lowest support, the classifier building process can be completed in a very short time, compared to the previous naive methods. If these two algorithms are applied, the use of associative classifiers becomes feasible for large-scale text classification. The proposed approach can also be applied to other types of data than text.

Acknowledgment

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program (NIPA-2012-(H0301-12-3001)) supervised by the NIPA (National IT Industry Promotion Agency).

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, & C. Zaniolo (Eds.), *Proc. 20th int. conf. very large data bases, VLDB* (pp. 487–499). Morgan Kaufman. 12–15. <citeseer.ist.psu.edu/agrawal94fast.html>.

Antonie, M.-L., & Zaïane, O. R. (2002). Text document categorization by term association. In *ICDM '02: Proceedings of the 2002 IEEE international conference on data mining* (pp. 19). Washington, DC, USA: IEEE Computer Society.

Apté, C., Damerau, F., & Weiss, S. M. (1994). Towards language independent automated learning of text categorisation models. In *SIGIR* (pp. 23–30).

Cheng, H., Yan, X., Han, J., & Hsu, C.-W. (2007). Discriminative frequent pattern analysis for effective classification. In *ICDE* (pp. 716–725).

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In W. Chen, J. Naughton, & P. A. Bernstein (Eds.), *2000 ACM SIGMOD intl. conference on management of data* (pp. 1–12). ACM Press. <citeseer.ist.psu.edu/han99mining.html>, 05.

Lang, K. (1995). NewsWeeder: Learning to filter netnews. In *Proceedings of the 12th international conference on machine learning* (pp. 331–339). San Mateo, CA, USA: Morgan Kaufmann publishers Inc. <citeseer.ist.psu.edu/lang95newsweeder.html>.

Li, B., Sugandh, N., Garcia, E. V., & Ram, A. (2007). Adapting associative classification to text categorization. In *DocEng '07: Proceedings of the 2007 ACM symposium on document engineering* (pp. 205–208). New York, NY, USA: ACM.

Li, W., Han, J., & Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple class-association rules. In *ICDM* (pp. 369–376). <citeseer.ist.psu.edu/li01cmar.html>.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. In *Knowledge discovery and data mining* (pp. 80–86). <citeseer.ist.psu.edu/liu98integrating.html>.

McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. <http://www.cs.cmu.edu/mccallum/bow>.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *ICDT '99: Proceedings of the 7th international conference on database theory* (pp. 398–416). London, UK: Springer-Verlag.

Sandhya, N., Lalitha, Y. S., Govardhan, D. A., & Anuradha, D. K. (2011). An improved approach for document retrieval using suffix trees. *IJACSA, 2*, 33–36.

She, R., Chen, F., Wang, K., & Ester, M. (2003). Frequent-subsequence-based prediction of outer membrane proteins. In *Proceedings of 2003 international conference on data mining and knowledge discovery* (pp. 436–445). ACM Press.

Wang, J., & Karypis, G. (2005). Harmony: Efficiently mining the best rules for classification. In *SDM*.

Weiner, P. (1973). Linear pattern matching algorithms. In *SWAT (FOCS)* (pp. 1–11).

Yoon, Y., & Lee, G. G. (2008). Text categorization based on boosting association rules. In *Proceedings of ICSC 2008, second IEEE international conference on semantic computing* (pp. 136–143).

Zaki, M. J. (2004). Mining non-redundant association rules. *Data Mining and Knowledge Discovery: An International Journal, 9*(3), 223–248.

Zobel, J., Moffat, A., & Ramamohanarao, K. (1998). Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems, 23*, 453–490.