BAYESIAN CLASSIFICATION AND FREQUENT PATTERN MINING FOR APPLYING INTRUSION DETECTION

Heon Gyu Lee, Kiyong Noh, Keun Ho Ryu
Database & Bioinformatics Laboratory, Chungbuk National University, Korea
Cheongju Chungbuk, 361-763, Korea
{hglee, khryu}@dblab.cbu.ac.kr

ABSTRACT:
In this paper, in order to identify and recognize attack patterns, we propose a Bayesian classification using frequent patterns. In theory, Bayesian classifiers guarantee the minimum error rate compared to all other classifiers. However, in practice this is not always the case owing to inaccuracies in the unrealistic assumption(class conditional independence) made for its use. Our method addresses the problem of attribute dependence by discovering frequent patterns. It generates frequent patterns using an efficient FP-growth approach. Since the volume of patterns produced can be large, we propose a pruning technique for selection only interesting patterns. Also, this method estimates the probability of a new case using different product approximations, where each product approximation assumes different independence of the attributes. Our experiments show that the proposed classifier achieves higher accuracy and is more efficient than other classifiers.
KEY WORDS: Intrusion Detection, Data Mining, Classification.

1. INTRODUCTION

Intrusion detection[1] aims for the detection of illegal activities. It attempts to analyse existing attack patterns and recognize new intrusion methods, employing methods from fields such as statistics, data mining and machine learning. The purpose of this paper is to investigate the effectiveness and accuracy of classification method for the intrusion detection. To achieve this purpose, we apply classification[2] and frequent pattern mining techniques[3],[4]. We analyse the existing intrusion detection techniques using data mining. Based on the results, we suggest a hybrid approach that attempts to utilize the advantages of both frequent pattern mining and Bayesian classification for the intrusion detection. The proposed classifier is Bayesian classifier based on frequent patterns and can relax independence assumption of classifiers like NB(Naive Bayesian) and DT(Decision Tree). For example, the NB[5] makes the assumption of conditional independence, that is, given the class label of a sample, the values of the attributes are conditionally independent of one another. When the assumption holds true, then the NB is the most accurate in comparison with all other classifiers. In practice, however, dependences can exist between variables of the real data. Our classifier can consider dependences of the audit data and relax the strong independence assumption implied by NB and uses the supports of frequent patterns to approximate probabilities. Frequent pattern is a set of non-redundant and interest patterns discovered in training phase. In classification phase, the probability of a new case can be estimated using different product approximations, where each product approximation assumes different independence of the attributes. In this paper we describe our main contributions as follows.

- FP-growth is extended to perform frequent patterns mining. We call the method CFP-growth and the corresponding data structure CFP-tree.
- CP-tree is used to prune redundant patterns.
- The proposed classifier applies the Bayesian to perform automatic audit data classification using product approximation.

2. FREQUENT PATTERNS DISCOVERY

2.1 CFP-tree Construction Algorithm

For construction of CFP(Class Frequent Pattern)-tree, the following are modifications to the FP-tree.
- Node class count is added in the CFP-tree, where each element of the distributions stores a number of transactions of the current class containing a pattern from this node to the root. Thus, node count value is sum of the each node class count.
- Every item class count value is added in the frequent-item header table of the CFP-tree. Thus, every item count value is sum of the each item class count.
- Every pattern inserted into the tree is sorted in ascending order of frequency to make the redundant pattern pruning possible.

For a pattern to be inserted into the CFP-tree, we require it to be a frequent pattern satisfying support threshold S_{min}. Let D be a database containing |D| task-relevant transactions and $C=\{C_1,\ldots,C_n\}$ be a finite set of class.

Definition 2.1 Class support and support of pattern A:

$$Class\ Support(A.\sup_i) = \frac{count(A, C_i)}{|D|} = P(A, C_i),$$

$$Support(A.\sup) = \frac{\sum_i count(A, C_i)}{|D|} = P(A)$$

Definition 2.2 CFP(Class Frequent Pattern)-tree: A class frequent pattern tree is an expanded prefix tree structure storing crucial quantitative information about FPs.

- The tree has a root labeled as null, a set of item prefix sub trees as the children of the root, and a frequent item header table.
- Each node in the item prefix sub tree contains three fields: item-id(name), item class count value for each class C_i and node-link.
- The number of entries in frequent item header table is equal to the number of distinct elements in the CFP-tree and each entry contains three fields: item-id(name), item class count value for each class C_i and head of node-link.

Let's examine an example that builds CFP-tree for the database in Table 2.1 with S_{min}=2 and the algorithm 2.1 describes the process of CFP-tree construction

- Scan the database once; collect the count for each item and eliminate those items whose support is not greater than S_{min}. After step 1, the list of frequent 1-itemsets is a_1:3, b_2:4, c_1:2, d_3:3 and L={c_1:2, a_1:3, d_3:3, b_2:4}.
- Scan the database from the start again. For each database, filter out the infrequent items and sort the remaining ones in the ascending order of frequency; Insert the pattern into the CFP-tree as a branch. Figure 2.1 shows a process of the CFP-tree construction.

Table 2.1 Transaction database

TID	Attributes				Class Label	Inserted Patterns
	A	B	C	D		
1	a_1	b_2	c_3	d_3	C1	{C1: a_1, d_3, b_2}
2	a_1	b_2	c_1	d_2	C2	{C2: c_1, a_1, b_2}
3	a_2	b_3	c_2	d_3	C1	{C1: d_3}
4	a_3	b_2	c_4	d_1	C2	{C2: b_2}
5	a_1	b_2	c_1	d_3	C2	{C2: c_1, a_1, d_3, b_2}

Scan D once and collect the set of frequent pattern F and their supports. Sort F in support ascending order as L, the list of frequent items.
Create the root R of a new CFP-tree and label it as "null"
Create frequent item header with |F| entries. Set all head of node-link pointer to null.
for each transaction T∈D do {
  C_T = class of transaction T;
  Select only frequent items of T into a record P;
  Sort P in the order of L. Call insert_tree(P, C_T, R);
 }

insert_tree(P, c, R)
 Let P=[p|P-p], where p is the first element of P, P-p is the remaining list.
 if R has a child N such that N.item-id=p then
    N.count(c) = N.count(c)+1;
    N.count=N.count+N.count(c);
 else {
     create a new node N;
     for all classes c_i do {
         if c_i=c then
           N.count(c_i)=1; N.count=N.count+N.count(c_i);
         else N.count(c_i)=0;
        N.item-id=p; N.parent=R;
        N.node-link=H(p).head; H(p).head=N; }
 }
 H(p).count = H(p).count+1;
 if P-p≠0 then Call insert_tree(P-p, c, N);

Algorithm 2.1 CFP-tree construction from database

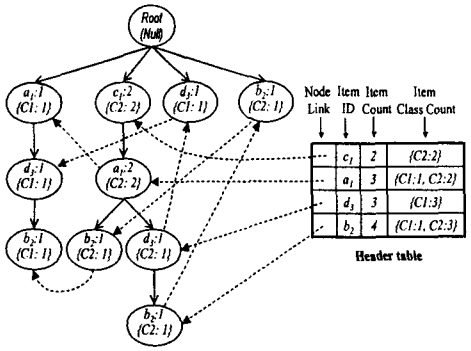

Figure 2.1 CFP-tree construction

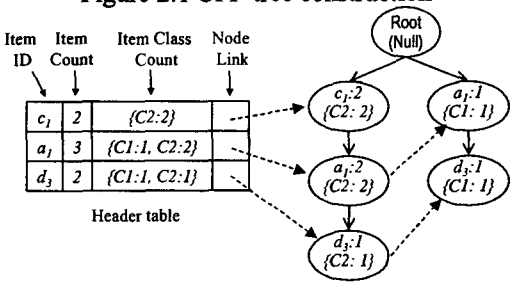

Figure 2.2 Conditional CFP-tree for item b_2

2.2 FPs Discovery using CFP-tree

The CFP-growth method recursively searches for shorter FPs and then finds longer patterns by concatenating a frequent item with a shorter frequent suffix pattern. We mine CFP-tree bottom-up. Starting from b_2, for each frequent 1-item, we construct its pattern base. A conditional pattern base for an item contains the transactions that end with that item. We then treat the conditional pattern base as a transaction and build a conditional CFP-tree. The CFP-growth is recursively performed on such conditional CFP-trees. Item b_2's conditional pattern base is: {(a_1:1(C1:1), d_3:1(C1:1)), (c_1:1(C2:1), a_1:1(C2:1)), (c_1:1(C2:1), a_1:1(C2:1), d_3:1(C2:1))}. Figure 2.2 shows a process of the conditional CFP-tree construction. There are two branches in the b_2's conditional CFP-tree. The possible combinations are: {c_1b_2:2(C2:2)}, {a_1b_2:3(C1:1,C2:2)}, {d_3b_2:2(C1:1,C2:1)}, {$c_1a_1b_2$:2(C2:2)}, {$a_1d_3b_2$:2(C1:1,C2:1)}. Item d_3's conditional pattern base is: {(a_1:1(C1:1)), (c_1:1(C2:1), a_1:1(C2:1))}. In this conditional pattern base c_1 occurs only once and thus is eliminated. The possible combination is only one pattern {a_1d_3:2(C1:1, C2:1)}. Item a_1's conditional pattern base is: {(c_1:2(C2:2))}. Similarly, we construct a_1's CFP-tree and generate the one FP as: {c_1a_1:2(C2:2)}. Lastly, combined with the frequent 1-items generated during the first database scan, we can get the set of all FPs and their class counts for each class.

2.3 Pruning Redundant Pattern using CP-tree

To remove redundant pattern, we use CP(Compressed Pattern)-tree. CP-tree stores support of FP and their class count. CP-tree contains also pattern-ranking criterion

such as support for patterns. Pattern ranking is needed to select the best pattern in case of overlapping patterns.

Before the pruning using CP-tree, all patterns are ranked according to a ranking criteria defined as follows.

Definition 2.3 Pattern-Ranking: Given two patterns, A_i and A_j, $A_i \gg A_j$ (or A_i is ranked higher than A_j) if one of the follow conditions is satisfied

(1) $Sup(A_i) > Sup(A_j)$, or

(2) $Sup(A_i) = Sup(A_j)$, A_i is generated earlier than A_j.

The FPs are inserted into the CP-tree and pruned in parallel. When inserting a new pattern A into the tree, the three cases can happen. First, the tree contains a sub pattern of A ranked higher than A. This pattern is redundant and is not inserted. Second, the tree contains one or more supper patterns of A ranked lower than A. In this case, all the supper patterns are pruned. Third, None of the above two case happens, and A is inserted the CP-tree.

```
create a root of CP-tree;
for each pattern p∈P do {
  if there is no p_sub of p, rank(p_sub)≫rank(p) then {
    if there is no p_supper of p, rank(p_supper)≪rank(p) then
        prune all p_sub;
    insert p into CP-Tree;
    }
}
```

Algorithm 2.2 Redundant pattern pruning using CFP-tree

3. BAYESIA CLASSIFICATION USING EFP_S

3.1 Rule pruning using the cover principle

Once a complete CP-tree has been produced the generated rules are placed into a list P, ordered according to the Pattern Ranking Criteria, which is then pruned. The pruning algorithm using the cover principle is presented in algorithm 3.1.

```
P = sort(FPs); // according to the their rank
for each data case d_i ⊆ D do {
    database cover count dcover_i=0;  EFP=ø;
}
for (the training dataset D and P ≠ ø) do {
    for each pattern p_i ⊆ P do {
        find a set Dcover⊆D of cases covered by p_i;
        if at least one d⊆Dcover is classified by p_i then {
            EFP = EFP ∪ p_i;
            for each case d_j ⊆ Dcover do {
                update dcover_j = dcover_j+1;
                if dcover_j = τ then delete case m_j from D;
            }
        }
    }
}
```

Algorithm 3.1 Database coverage pruning

3.2 BCFP Algorithm for Classification

When a new case $A'=\{a_1, a_2, \ldots, a_n\}$ arrives to be classified, BCFP(Bayesin Classifier using Frequent Pattern) combines the evidence provided by the subsets of A' that are presented in EFP(Efficient Frequent Pattern) to approximate $P(A', C_i)$, where EFP denotes the final high quality FPs for classification and $P(A', C_i)$ determines the conditional probability $P(C_i|A')$.

Definition 3.1 A set B with respect to case A': $B=\{f \in EFP | f \subset A'\}$. The B consists of the longest possible patterns of EFPs that are subsets of A'.

BCFP uses the EFPs of B to derive product approximations of $P(A', C_i)$ for all classes. The product approximation of the probability of an n-itemset A' contains a sequence of at most n subsets of A' such that each pattern contains at least one item not covered in previous patterns. The chain rule is $P(a_1, a_2, \ldots, a_n) = P(a_1)P(a_2|a_1)\ldots P(a_n|a_1,\ldots,a_{n-1})$. To obtain a product approximation of $P(A', C_i)$, the patterns are combined using the chain rule of probability while under the assumption that all necessary attributes are independent. For example, suppose a test case $A'=\{a_1, a_2, \ldots, a_5\}$ arrives. From the EFPs, we find $B=\{(a_2, a_5), (a_3, a_4), (a_1, a_2, a_3), (a_1, a_4, a_5)\}$ corresponding to A.

Different combinations of B lead to different product approximation, and the product approximations are different even when the same patterns are used in different order. The product approximation of $P(A', C_i)$ is generated by adding one pattern at a time till no more patterns can be added, that is when either all the items of the remaining patterns from B are already covered or no more patterns are available in B. For constructing product approximation, patterns of B are first sorted according to pattern ranking criteria of definition 2.3, and then essential patterns are selected from the beginning to construct the product approximation. The set of covered items is denoted as $item_{cov}$. A pattern p inserted in the product approximation satisfies the following definition.

Definition 3.2 Given two patterns p, q and the set of covered items $item_{cov}$.

Rule 1: $|p\text{-}item_{cov}| \geq 1$, Rule 2: $length(p) < length(q)$, Rule 3: $|p\text{-}item_{cov}| \leq |q\text{-}item_{cov}|$

Rule 1 means p contains new items that have not been covered. And it guarantees the product solution satisfies the chain rule. Rule 2 assures shorter patterns be considered first. This is equivalent to maximizing the number of patterns used in the product approximation. Rule 3 gives priority to those patterns among the remaining alternatives that contain the smallest number of not covered items. The algorithm 3.2 for Bayesian classification using EFPs is given below and this algorithm first finds the evidence B provided by the subsets of A' that are present in EFPs. Cov is the subset of A' already covered, Num and Den are the sets of patterns in numerator and denominator, respectively. Procedure selectNext() extracts from B the next pattern to be used in the product approximation. The algorithm stops once all items in A' have been covered.

```
B={f∈EFP | f⊂A'};  Cov=ø;  Nom=ø Den=ø
for (i=1;Cov⊂A';i++) do {
    B_i = selectNext(Cov, B);
```

Num= Num $\cup B_i$;
Den=Den $\cup (B_i \cap$ Cov); Cov=Cov $\cup B_i$; }
for each class C_i do {
$P(A', C_i) = P(C_i) \prod_{a \in Num} P(a, C_i) / \prod_{b \in Den} P(b, C_i)$; }
The class C_i with maximal P(A', C_i);
Procedure selectNext(Cov, B)
S={p$\in$B$\wedge$|p-Cov|$\geq$ 1};
return a pattern $B_i \in S$
such that for all other patterns $B_j \in S$
length(B_i) < length(B_j); length(B_i) = length(B_j) and |B_i-Cov|$\leq$ |B_j-Cov|;
Algorithm 3.2 Bayesian classification using Frequent Patterns

4. EXPERIMENTS AND RESULTS

We evaluate our experiments in building intrusion detection model on the dataset from the KDD'99[6]. Our experiments focus Dos and Probe attacks. To accurate attack detection, we extract different features that reflect characteristics of each attack type. Since the extracted dataset contains continuous variables, those variables must be made discrete. Therefore, decision tree[7] has been used because the intervals are selected according to the information they contribute target variable. To evaluate the performance of our algorithms, first we compared CFP-growth with previous ones. The previous data mining results were using Apriori method[2].

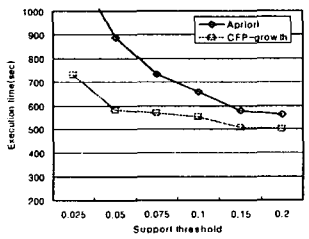

Figure 4.1 Mining set of FPs

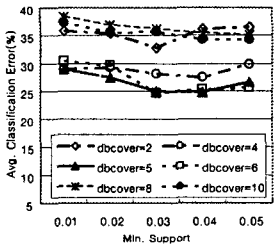

Figure. 4.2 Accuracy on data

We conclude that between these algorithms, CFP-growth takes less time to generate the complete set of frequent patterns. We have two important thresholds(support and coverage) for the classifier performance. These thresholds control the number of FP selected for constructing classifier. Figure 4.2 shows average classification error according to minimum support and database coverage respectively. Average classification error is a fraction of misclassified connection records, where a record is considered misclassified if the highest probability class predicted is different from a correct one.

In Figure 4.3, the experiment shows that in terms of accuracy BCFP and CMAR[8]. NB has lower accuracy in two cases. Although BCFP was not more accurate and efficient than CMAR, we are satisfied with these experiments because BCFP showed efficient classifier construction time and was more accurate than NB that makes the assumption of conditional independence.

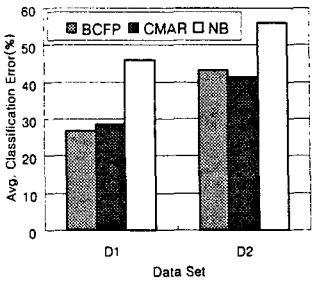

Figure. 4.3 Comparison of BCFP, CMAR, and NB accuracy

5. CONCLUSION

The purpose of this paper was to develop accurate mining algorithm to automatically audit data classification. The proposed algorithm has been implemented by combination of two data mining techniques, frequent pattern discovery and Bayesian classification, and which uses FPs to construct different product approximations. For achieving the high efficiency of FP-growth, we introduced a CFP-growth algorithm that extends FP-growth by using CP-tree for redundant pattern pruning. Experimental results showed high accuracy and efficiency achieved by our classifier. This classifier was compared with both models of NB and CMAR classifier. It was shown to outperform both NB and CMAR in accuracy.

ACKNOWLEDGEMENTS

This work was supported by the RRC program of MOCIE and ITEP.

REFERENCES

[1] W. Lee, S. J. Stolfo and K. W. Mok, "A Data Mining Framework for Building Intrusion Detection Models", In Proc. of the IEEE Symposium Security and Privacy,120-132.

[2] Chris Sinclair, Lyn Pierce and Sara Matzner, "An Application of Machine Learning to Network Intrusion Detection", 15th Annual Computer Security Applications Conference December. Phoenix, Arizona (1999)

[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", In Proc. of the 20th VLDB Conference, Santiago, Chile (1994) 487-499

[4] J. Han, J. Pei and Y. Yin, "Mining frequent patterns without candidate generation", In SIGMOD'00, May (2000)

[5] R. Duda and P. Hart "Pattern classification and scene analysis," John Wiley and Sons, New York, (1973)

[6] KDD Cup 1999 Data Set, used for The Third International Knowledge Discovery and Data Mining Tools Competition, one of several possible homepage URIs: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[7] U. M. Fayyad and K. B. Irani, "Multi-Interval discretization of continuous-valued attributes for classification learning", In Proc. of the 13th International Joint Conference on Artificial Intelligence (1993) 1022-1027

[8] W. Li, J. Han and J. Pei, "CMAR: Accurate and Efficient Classification Based on Multiple Association Rules", In Proc. of 2001 International Conference on Data Mining (2001)