

# CBS: A New Classification Method by Using Sequential Patterns

Vincent S. M. Tseng Chao-Hui Lee

Dept. Computer Science and Information Engineering  
National Cheng Kung University  
Tainan, Taiwan, R.O.C.  
Email: [tsengsm@mail.ncku.edu.tw](mailto:tsengsm@mail.ncku.edu.tw)

**Abstract** - Data classification is an important topic in data mining field due to the wide applications. A number of related methods have been proposed based on the well-known learning models like decision tree or neural network. However, these kinds of classification methods may not perform well in mining time sequence datasets like time-series gene expression data. In this paper, we propose a new data mining method, namely Classify-By-Sequence (CBS), for classifying large time-series datasets. The main methodology of CBS method is to integrate the sequential pattern mining with the probabilistic induction such that the inherent sequential patterns can be extracted efficiently and the classification task be done more accurately. Meanwhile, CBS method has the merit of simplicity in implementation. Through experimental evaluation, the CBS method is shown to outperform other methods greatly in the classification accuracy.

**Keywords:** Classification, Sequential Pattern, Time-series data, Data Mining.

## 1. Introduction

In recent years, many data mining techniques emerged in various research topics like association rules, sequential patterns, clustering and classification [1, 2, 4, 11, 12]. These techniques are also widely used in different application fields. Most of the existing data mining methods are designed for solving some specific problem independently. In the other hand, some few compound methods integrate two or more kinds of data mining techniques to solve complex problems. For example, *Classification By Association rules (CBA)* [8] and other integrated methods like the SPF Classifier in [7] are of this kind. These compound methods can effectively utilize the advantages of each individual mining technique to improve the overall performance in data mining tasks. For example, the CBA [8] method was reported to deliver higher accuracy than traditional classification method like C4.5 [11]. Hence, it is a promising direction to integrate different kinds of data mining methods to form a new methodology for solving complex data mining problems.

Among the rich data mining problems, classification modeling and prediction is an important one due to the wide applications. Numerous classification methods have been proposed [3] [4] [5] [6] [8] [9] [11], including the most popular ones like decision tree, neural network and support vector machine. The goal of a classification method is to build up a model that can relate the label (class) to the values of attributes for the data instances. Hence, the label of a new data instance can be predicted correctly by using the built model. However, few studies explore the issue of using integrated data mining methods to classify datasets with time sequences, e.g., the time-series gene expression datasets.

One of the few related work is the one by Lesh *et al.* [7], which combined the sequential patterns mining and tradition classification method like C4.5 to classify sequence datasets. In this paper, we present a new method named *Classify-By-Sequence (CBS)*, which integrates the sequential pattern mining with the probabilistic induction such that the inherent sequential patterns can be extracted and used for classification efficiently. The experimental results show that the CBS method achieves higher accuracy in classification than other methods.

## 2. Problem Definition

Consider a large database that stores numerous data records. Each data record is consisted of a data sequence and associated with a class label. The main problem is to build a classifier based on the information in the database such that the class of a new data instance can be predicted correctly. Let  $D_i$  represents all time-series data instances belonging to class  $i$ . So, the database is represented as  $D = \{D_1, D_2, D_3, D_4, \dots, D_N\}$  assuming there are totally  $N$  classes in the database. For each class, data set  $D_i$  consists of time-series data instances in the form of  $\{a_1, a_2, a_3, \dots, a_n\}$ , where  $a_n$  represents the value at  $n$ th time point. To simplify this study, it is assumed that the value at each time point is discretized and transformed into categorical values in advance.

Our goal is to discover the *Classifiable Sequential Patterns (CSP)* rules that can be used as the classification rules for the classifier. The CSP rule is in the form of  $SP_i \rightarrow C_m$ , where  $SP_i$  is a sequence like  $a_2 \rightarrow a_3 \rightarrow a_7, \dots$ , and

$C_m$  is some class  $m$ . We will describe in details how to discover these CSP rules in next section.

### 3. The CBS Method

With the basic idea of CBS as described previously, we propose two variations of CBS methods, namely CBS\_ALL and CBS\_CLASS, for discovering the CSP rules. After the CSP rules are discovered, it is easy to do the class prediction by using the induction approach. Hence, we will focus on the procedures of mining the CSP rules in the following.

#### 3.1 CBS\_ALL Method

The concept of this method is similar to the sequential patterns mining with the add-in of probabilistic induction. First, we extract all classifiable sequences from the database by using some sequential patterns mining. In addition to finding the frequent sequences, we further give the *classify-score* for these sequences. CBS\_ALL algorithm considers the *class support* and *transaction support* at the same time. We adopt an Apriori-like procedure for implementing the support counting task. Figure 1 shows the CBS\_ALL algorithm in details. Thus, we can get a number of CSP rules. Each CSP rule contains the classification information and we use all CSP rules to establish a classifier. Figure 2 illustrates the policy of the classifier in using all CSP rules to classify a time-series data correctly.

Notice that this algorithm deals with the categorical time-series data only since it is assumed that all original datum are transformed into categorical values as mentioned in Section 2. In CBS\_ALL algorithm, we extract large-1 items as  $CSP_1$ , which are further used to generate  $SP_2$  (candidate large-2 Sequential Patterns). In counting the support, if SP is a subsequence of a transaction sequence data,  $SP.seq\_sup$  (i.e., support of the sequence) will be incremented. Another important procedure for producing classify-score of a sequences is counting the support of each class. In our design, both SP and CSP are given a *class\_sup* array, respectively. In counting sequence support, the class support is counted at the same time. If the SP is with the class label  $C_x$ , then  $SP.class\_sup[x]$  will also be incremented. After the whole dataset has been counted for SP, we prune SP into CSP with their *seq\_sup* and *class\_sup[]*. In performing the pruning, a rule is reserved if at least one class support is larger than the minimum rule support; otherwise it is eliminated since the sequential pattern distributes in too many classes and is unqualified as a rule. Then, the procedure loops back to the candidate generation procedure. The algorithm generates all CSP iteratively

until no more SP meets the requirements of *min\_seq\_sup* and *min\_class\_sup* thresholds.

```

CBS _ ALL (Dataset D, min_seq_s up, min_rule_s up)
{
  CSP1 = {large 1 - items}
  for(i = 2; CSPi ≠ φ; i++) do
    SPi = gen _ candidateS P(CSPi-1);
    for each data d ∈ D do
      SPs = SPi ∩ subseq(d);
      for each sp ∈ SPs do
        sp.seq_ sup++;
        sp.class_ sup[d.class ]++;
      end
    end
    CSPi = {sp/sp ∈ SPi, sp.seq_ sup ≥ min_seq_s up
and ∃w let sp.class_ sup[w] / sp.seq_ sup ≥ min_rule_s up}
  end
  CSP = ∪i CSPi
}

```

Figure 1. CBS\_ALL algorithm

For the classification part, we use a scoring method to determine the class label for a newly given data instance with the data sequence by utilizing the CSP rules as described above. If the subsequence of a sequence  $x$  equals to the SP of CSP  $y$ , we call that the sequence  $x$  matches CSP  $y$ . Figure 2 shows the procedure in details.

First, we find out all CSP with sequences matching the subsequences of the sequence in the new data instance. We use the class support and sequence support to calculate the score for each selected CSP. In this way, we obtain the scores for each class by their induced probability. Finally, the new data instance is assigned with the class of the highest score. Although this classification task is done through a simple score counting process, the algorithm takes into account two important factors – the length of CSP and the subsequence patterns for the matched CSP. As an easy example, assume a CSP A is the subsequence of CSP B. Consequently, if a new sequence contains CSP B, it must also contain CSP A. This means that we may incur the problem of repeated counts in calculating some scores. To resolve this problem, it seems that we should remove all matched sequences like CSP A. However, this problem is eliminated by considering another factor, namely the size of the matched CSP. By our scoring policy, the CSP is weighted according to their sequence length. In this way, for example, the CSP with length 5 get higher weights than the CSP with length 1. In fact, we have tried other measures for the weighting, like the product of length and the original score, but they did not produce better results. Hence, we use the subsequence relation as weight for the CSP.

```

Class_of_sequence (sequence x)
{
  M =  $\phi$ ;
  for each cspi ∈ CSP do
    if cspi.sp ∈ subseq(x)
      M.add(cspi);
  end
  score_array[] = new array[class_set(D).count];
  for each cspm ∈ M do
    for each cn ∈ class_set(D) do
      score_array[n] +=
cspm.support / cspm.class_support[n];
    end
  end
  k = indexof (Max{score_array[]});
  return k;
}

```

**Figure 2. CBS\_ALL classifier**

```

class_of_sequence(sequence x)
{
  total_score[] = new array[class_set(D)];
  for each cspi ∈ CSP do
    total_score[cspi.class] += cspi.sp.length;
  end
  score_array[] = new array[class_set(D)];
  for each cspi ∈ CSP do
    if cspi ∈ subseq(x)
      for each cm ∈ belong_classes_set(cspi)
        score_array[m] +=
cspi.sp.length / total_score[m];
      end
    end if
  end
  k = index_of(Max{score_array[]});
  return ck
}

```

**Figure 3. CBS\_CLASS algorithm**

### 3.2 CBS\_CLASS Method

This method separates the database into groups by class labels. Different from CBS\_ALL, the classifiable sequences are extracted from each class group, respectively, rather than from the whole dataset. Similarly, the classifier is built by using the extracted sequences. The concept of this method is to focus on the features of each class group in generating the classification rules. Figure 3 shows the detail of the CBS\_CLASS algorithm.

Different from CBS\_ALL algorithm, only one parameter, the minimum support, is needed in CBS\_CLASS. The procedure FindSP in Figure 3 adopts an Apriori-like approach for mining sequential patterns. After the sequential patterns are extracted from each class, we can use these sequential patterns to classify a new sequence data instance directly. Figure 4 shows the classifier algorithm for CBS\_CLASS, which also uses the

scoring approach to classify new data instance, but there are no for sequence support score and class support score in CBS\_CLASS. The procedure of classifier is similar to that of CBS\_ALL, with some differences in the score counting for each CSP. In CBS\_CLASS classifier algorithm, we take the sequence length as the CSP score. Then, we normalize the total scores of each class into the same value base such that the maximum score for a new sequence in each class is 1.

Intuitively, the CBS-CLASS method is more effective on sequence feature mining. It not only eliminates the factor of data quantity imbalance between classes but also extracts the real sequential patterns for each class sequence data. In contrast, the CSPs of CBS\_ALL are frequent sequences in terms of the whole dataset. Hence, they may not be deterministic features for classification.

```

CBS_CLASS( Dataset D, min_sup)
{
  for each ci ∈ class_set(D) do
    Di = class_dataset(D, ci);
    CSPi = FindSP(Di, min_sup);
  }
  FindSP(Dataset D, min_sup)
  {
    SP1 = {large 1-items}
    for( i = 2; SPi-1 ≠  $\phi$ ; i++) do
      SPCi = gen_candidateSP(SPi-1);
      for each data d ∈ D do
        SPs = SPCi ∩ subseq(d);
        for each sp ∈ SPs do
          sp.support++;
        end
      end
      SPi = {sp | sp ∈ SPs, sp.support ≥ min_sup}
    end
    return  $\bigcup_i SP_i$ 
  }
}

```

**Figure 4. CBS\_CLASS classifier**

## 4. Experimental Evaluation

In the following, we describe the experimental results in evaluating the performance of CBS-ALL, CBS-CLASS and other method by using the simulated time-series datasets.

### 4.1 Synthetic Data Generator

We design a synthetic data generator that can generate time-series datasets with different properties based on the parameters as listed in Table 1. The default value for each parameter in the following experiments is also given in Table 1.

### 4.2 Comparisons of CBS\_ALL and CBS\_CLASS

Figure 5 shows the result of the first experimental, in which the parameter *pattern\_len* is varied from 3 to 7 with other parameters as set in Table 1. Both of the inner test

and outer test results are given for the CBS\_ALL and CBS\_CLASS algorithms. We can see CBS\_CLASS performs much better than CBS\_ALL in classification accuracy. This result is due to the fact that CBS\_CLASS can extract more precise sequential patterns than CBS\_ALL for performing the classification.

Figure 6 shows the result of the second experiment in which the parameter *pattern\_count* is varied from 3 to 7. It is observed that the accuracy of both algorithms goes down when the number of hidden pattern is increased. This is because that, with more hidden patterns, the time-series data in each class becomes more diverse and hard to extract. This result matches our inference. We also get the similar observation that CBS\_CLASS achieves much higher accuracy than CBS\_ALL. This indicates that CBS\_CLASS is more stable than CBS\_ALL when the dataset is more complex in terms of the hidden patterns.

The above observations support our intuitive induction that CBS\_CLASS is better than CBS\_ALL method for sequence classification due to its property in isolating each class data during the mining of CSP rules. From algorithm viewpoint, the CBS\_CLASS method makes the data mining procedure more stable and powerful by processing each class data set separately.

Table 1. Parameters for the synthetic data generator.

Parameter	Description	Value
<i>seq_len</i>	Number of items in each time sequence	10
<i>pattern_len</i>	Length of hidden sequential patterns	5
<i>value_level</i>	The discretized level of value	100
<i>seq_count</i>	Number of time sequences	5000
<i>class_count</i>	Number of classes	10
<i>pattern_count</i>	Number of hidden sequential pattern	5
<i>skew_ratio</i>	The degree of skew in quantity of classes	0

### 4.3 Comparisons with SPF Classifier

In this experiment, we compare the performance of CBS\_CLASS with the SPF-classifier (Sequential Pattern Feature classifier) proposed in [7]. We define the *improvement ratio*  $P(l)$  and *average improvement ratio*  $AVG(P)$  as follows:

$$P(l) = \frac{Accuracy(CBS\_CLASS(l)) - Accuracy(SPF(l))}{Accuracy(SPF(l))}$$

$l$ : *pattern\_len*

$$AVG(P) = \frac{\sum_{l=p_0}^{p_n} P(l)}{n}$$

Figure 7 shows the result of the first experiment. CBS\_CLASS presents about 25% improvement over SPF-classifier in average under different settings of *pattern\_len* as shown in Table 2. Figure 8 shows the performance of

both methods by varying *pattern\_count*. SPF-classifier delivers stable accuracy over different *pattern\_count*, while CBS\_CLASS presents lower accuracy with *pattern\_count* increased. But CBS\_CLASS outperform SPF in accuracy with about 24% of improvement in average for the outer test results.

Finally, we investigate the impact of varying parameter *skew\_ratio*, which controls the degree of skew in distribution of sequence classes. Figure 9 shows that both algorithms perform stable and even slightly better with *skew\_ratio* increased. This shows that CBS\_CLASS and SPF are insensitive to the distribution of sequence classes.

From the above experimental results, it is concluded that CBS\_CLASS delivers much better accuracy than SPF. Hence, the CBS\_CLASS method is verified to be promising in resolving the time-series data classification problem.

Table 2. improvement ratio for Figure 7.

	3	4	5	6	7	AVG
<i>improvement ratio(%)</i>	2.63	23.92	28.74	38.68	33.07	25.41

## 5. Concluding Remarks

We have presented a new method named CBS with two variations for classifying large time-series datasets. Through experimental evaluation, it is shown that CBS can classify time-series data with good accuracy by utilizing sequential patterns hidden in the datasets. It is shown that CBS\_CLASS always outperforms CBS\_ALL since the former builds up the classification model separately for each class. In comparisons with SPF-Classifier, CBS\_CLASS presents much higher accuracy under varied kinds of datasets although it is not as stable as SPF-Classifier. Meanwhile, CBS has the advantage that it is easy to implement with excellent execution performance. Hence, we believe CBS is a promising method for classifying time-series data in large scale.

In the future, we will extend the CBS method such that it can handle the time-series dataset with numerical values. Meanwhile, we will consider the problem of skewed data distribution and missing values. Another important research issue will be the effective pruning of produced CSP rules with the aim to improve the efficiency in the process of class prediction.

## References

- [1] R. Agrawal and R. Srikant, Fast Algorithms for Mining Association Rules, *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, September 1994.

[2] R. Agrawal and R. Srikant, Mining Sequential Patterns, *Proc. of the 11th Int'l Conference on Data Engineering*, Taiwan, March 1995.

[3] K. Ali, S. Manganaris, R. Srikant, Partial Classification using Association Rules, *Proc. of the 3rd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Newport Beach, California, August, 1997.

[4] R. J. Bayardo Jr. Brute-Force Mining of High-Confidence Classification Rules. *Proc. of the Third International Conference on Knowledge Discovery and Data Mining*, pp. 123-126, 1997.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees, *Wadsworth Int. Group*, Belmont, California, USA, 1984

[6] U. M. Fayyad and K. B. Irani, Multi-interval discretization of continuous valued attributes for classification learning. In R. Bajcsy (Ed.), *Proc. of the 13 International Joint Conference on Artificial Intelligence*, pp. 1022-1027, Morgan Kaufmann, 1993.

[7] N. Lesh, M. J. Zaki, M. Ogihara, Mining features for Sequence Classification, *5th ACM SIGKDD International*

*Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 342-246, San Diego, CA, August 1999.

[8] B. Liu, W. Hsu, Y. Ma, Integrating Classification and Association Rule Mining, *Proc. of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98, full paper)*, New York, USA, 1998.

[9] B. Liu, W. Hsu and S. Chen, Using General Impressions to Analyze Discovered Classification Rules, *Proc. of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97, full paper)*, pp. 31-36, August 14-17, Newport Beach, California, USA, 1997.

[10] M. Mehta, R. Agrawal and J. Rissanen, SLIQ: A Fast Scalable Classifier for Data Mining, *Proc. of the Fifth Int'l Conference on Extending Database Technology*, Avignon, France, March 1996.

[11] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufman, 1992.

[12] M. J. Zaki, Efficient Enumeration of Frequent Sequences, *7th International Conference on Information and Knowledge Management*, pp. 68-75, Washington DC, November 1998.

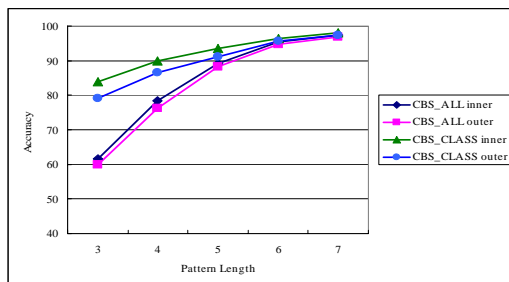


Figure 5. Comparative results by varying *pattern\_len*.

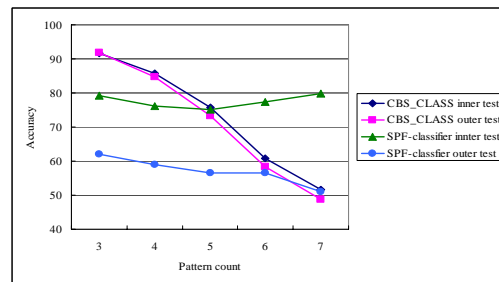


Figure 8. Comparative results by varying *pattern\_count*.

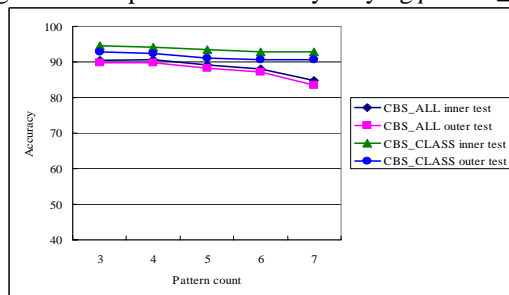


Figure 6. Comparative results by varying *pattern\_count*.

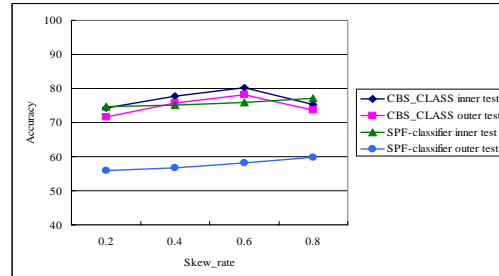


Figure 9. Comparative results by varying *data\_skew*.

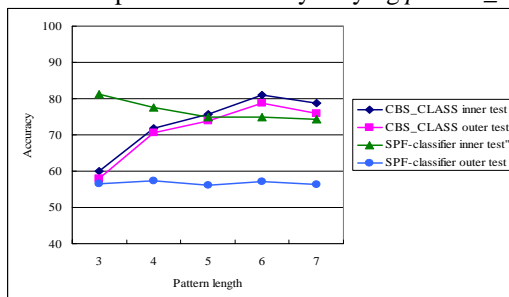


Figure 7. Comparative results by varying *pattern\_len*.