Knowledge Maintenance on Data Streams with Concept Drifting

Juggapong Natwichai and Xue Li

School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, Australia
{jpn, xueli}@itee.uq.edu.au

Abstract. Concept drifting in data streams often occurs unpredictably at any time. Currently many classification mining algorithms deal with this problem by using an incremental learning approach or ensemble classifiers approach. However, both of them can not make a prediction at any time exactly. In this paper, we propose a novel strategy for the maintenance of knowledge. Our approach stores and maintains knowledge in ambiguous decision table with current statistical indicators. With our disambiguation algorithm, a decision tree without any time problem can be synthesized on the fly efficiently. Our experiment results have shown that the accuracy rate of our approach is higher and smoother than other approaches. So, our algorithm is demonstrated to be a real anytime approach.

1 Introduction

There are many classification applications that require mining on the data streams, such as network sensor monitoring, stock market analysis or server performance tuning. Concept drifting [1] always occurs in streaming data environment because data is generated continuously. In general, concept drifting happens when the knowledge discovered in the past, is not applicable to the current incoming data/events any more, because of the inherent domain changes. More general, any previous truth is no longer valid in the current time.

In dealing with the concept drifting problem, there are many available algorithms so far. They can be categorized into two groups: increment learning approach [2–4], and ensemble classifiers algorithms [5–7]. Most algorithms provide knowledge for the users in form of decision trees [8] representation. However, these algorithms also have the same problem which is any time prediction. More specifically, we do not know what time period the users might have interests to predict from a decision tree. Both groups of algorithms only reflect to new coming knowledge and manipulate it to existing knowledge. Intuitively, this procedure should avoid noise, so it will take some time to justify the new knowledge whether it is noise or not. And, within the time, a decision tree becomes unstable and produces very high error rate. Apparently, this problem can be seen very clearly in incremental classifier. As shown in Figure 1 from CVFDT [2] approach,

J. Zhang, J.-H. He, and Y. Fu (Eds.): CIS 2004, LNCS 3314, pp. 705–710, 2004.

which we shaded the areas that represent the periods of unstable decision tree. There is not much chance for the users to get a right tree, particularly when justifying period is larger than the size of concept. On the other hand, if an approach of ensemble classifiers is used, it would take time to adjust weight between each classifier to keep track of concept drifting too.

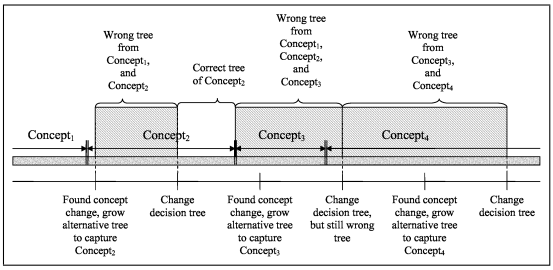

Fig. 1. Not Any Time Problem of Incremental Classifier

In this paper, we propose an approach for knowledge maintenance. Our approach has two phases. Firstly, we maintain knowledge by using an ambiguous Decision Table, (aDT) [9] as an intermediate form of knowledge representation. For reflecting changes, it is essential for counting accuracy of each rule in the decision table and also updating it constantly. So, we also add a set of indicators that is recorded for solving any time problem. This includes current error rate and time stamp for each rule. With ambiguous Decision Table (aDT), ambiguous rules will be kept for handling concept drifting problem. And, for making knowledge as recently as possible, we also implemented sliding window. Secondly, when the users want to predict based on acquired knowledge, we are able to generate a decision tree at any time using simple disambiguation algorithm from aDT.

The remaining sections in this paper are organized as following. In the next section, we introduce basic background related to our work. Section 3, gives the detail about our disambiguation algorithm. The experiment results are presented in section 4. Finally, section 5 concludes this papers and gives an overview of future work.

2 Ambiguous Decision Table with Time Stamp

In this section, aDT [9] and our extension work are reviewed. The aDT consists of two-dimensional array of cells. Each row records a decision rule, with each column within it is an attribute. And, the last column of each rule is an assigned class. Ambiguous decision table will contain two or more rules that conflict with each other e.g. in Table 1 rule 1 and 9, or rule 2 and 10. Our idea is to keep all happened knowledge, and add statistical indicators for each rule to make it ready for decision trees induction. So, we choose CVFDT algorithm [2] that can reflect concept drifting by generating alternative sub-tree to fill in this table.

When a new set of sub-tree is discovered by knowledge feeder, we will transform it into decision rules.

Furthermore, we also add two more columns into the aDT as shown in Table 1.

Table 1. Extended ambiguous Decision Table

Rule No.	Att_0	Att_1	Att_2	Att_3	Att_4	Class	% Error	Time Stamp
1	True	True	False	False	True	True	6	1
2	True	False	False	False	False	False	5	1
...	...	...	...	...	...	...	...	...
9	True	True	False	False	True	False	0	5
10	True	False	True	False	False	False	0	5

In the first, error of corresponding rule is recorded. And, start time stamp is also added here to support our disambiguation algorithm.

While incoming data keeps changing, new knowledge/rule will be derived. We keep making them always up-to-date by sliding window. Consequently, our extended aDT is capable to reflect any discovered knowledge as well as the history.

3 Decision Tree Synthesizing

In this section, we review the second phase of our approach. In the first phase, we collected knowledge along with its history and statistical indicators. When the users want to get the most recently knowledge so far, with additional algorithm we can synthesize decision tree for the users even we do not keep any decision tree. Because of simplicity of our algorithm, synthesizing process can be done on the fly.

So, we presented disambiguation algorithm as Table 2. At the beginning, we have to build the root of a decision tree. From the root of a decision tree to the lowest internal node, we will select all possible candidate attribute at currently state of aDT. All candidate attribute comes from each alternative sub-tree of our feeder. After any selection, we will split the tree by the number of attribute values. Each attribute value will have to be used to expand as a next level node of decision tree. And, this process will be repeated recursively. Because we maintain our decision table as an ambiguous table, we will also face ambiguousness in the synthesizing process. In this paper, we propose to induce all possible options firstly. And then, we eliminate those options with larger error when we induce until we reach a leaf node. This comes from the fact that the accuracy of decision trees come from where the decisions take place. At that point, we sum its error rate up for counting accuracy. The selected attribute at any level of a decision tree will be better than other candidates statistically. No matter whether it is newer than others or it is the oldest one, if it has less error rate than the other attributes at the current time, it would be selected. Although we keep all knowledge within the table, space complexity of our approach is $O(avc)$ where a

Table 2. Disambiguation Algorithm

Inputs:	aDT	is an extended ambiguous Decision Table,
	i	is a level number.
Outputs:	DT	is a decision tree,
	$DT.error$	is a error rate of decision tree.

Procedure Disambiguation(aDT**,** i**)**
Get candidate attributes for level ith from aDT.
While The number of candidates > 0 **do**
  split DT on each candidate attribute.
  For each child of DT on current split **do**
    refine aDT with each attribute value.
    If aDT has only one rule, **then**
      assign Class to current node of DT.
      Return DT.
    End if.
    Get $DT.error$ from **Disambiguation(**aDT**,** $i+1$**)**.$error$
  End for.
End while.
Choose DT from candidate with the least $DT.error$ and current level node error.
Return DT.

is the number of attributes, v is the maximum number of possible attribute values for any attribute, and c is the number of classes. This can be seen apparently that there is no term related to the number of examples. Moreover, we can assume that real-world streaming data is as large as multi-million examples and is produced continuously. So this asymptotic bound $O(avc)$ is not comparable with size of data streams and our approach can be used without causing efficiency degrading.

4 Experiment Results

We performed experiment using syntactic data set to demonstrate aspects of our algorithm when encountered with concept drifting. We selected hyperplane rotation in a d-dimension problem. A hyperplane in a d-dimensional space is denoted as Equation 1:

$$\sum_{i=1}^{n} a_i x_i = a_0 \tag{1}$$

where x_i is the coordinate of the ith dimension, and a_i is the weight corresponding to each ith dimension. For example, if $\sum_{i=1}^{n} a_i x_i \geq a0$, we will label

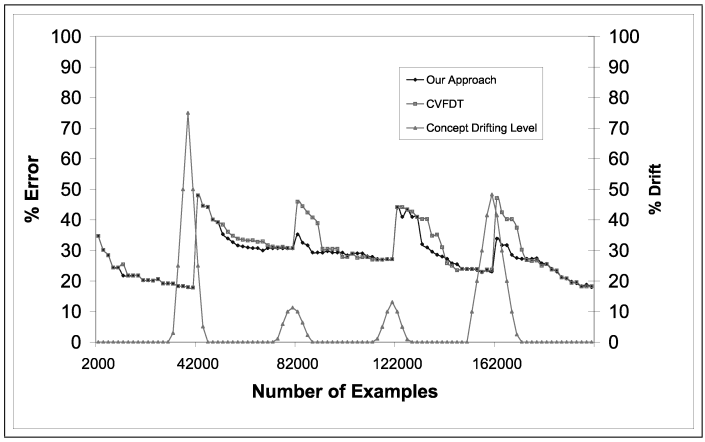

Fig. 2. Error Rate of Syntactic Data Set with Five Concepts

its class as positive. Otherwise it will be labelled as negative. We will also control the values of class distribution, so that both positive class, and negative class will be in the similar amount. The hyperplane rotation problem is an ideal case to demonstrate the handling of concept drifting, because it can adjust the weight of each attribute more smoothly than other ways (see [2] for more explanations).

We performed the experiment by generating 200,000 examples with 50 attributes syntactically. Four concept drifts were generated. We did this through adjustment of weight (a_i) of each attribute. In this way, we obtained five equal sets of syntactic data in this experiment.

Firstly, we made a change radically between Concept One and Concept Two. However, we adjusted weight between Concept Two and Three as smooth as possible. In addition, we wanted to investigate the behavior of our approach when encountered with a concept which was learned in the past. So, we could see efficiency of our approach to exploit previous knowledge. Concept drifting level along data set is shown as a minor vertical axis in Figure 2 .

The result of our approach is shown in Figure 2. Obviously, it is able to see that our approach is better than compared approach. The error-rate of our algorithm dropped sharply when it had learnt a new concept. If there is radical concept drifting, our approach would give dropping of error rate quickly as seen from error rate of Concept Two. And, in the case of slightly changed, concept catching period between Concept Two and Concept Three showed our better performance very clearly. In case of learning a concept which happened in the past, our algorithm can use it to synthesize a decision tree very efficiently too. So, at any time which the users need a decision tree, not much difference accuracy of the decision tree they will get.

5 Conclusion

This paper introduced a novel approach for solving concept drifting problem in mining knowledge from streaming data. With a data structure namely extended ambiguous Decision Table (aDT) [9], we can maintain time-stamped knowledge and handle the problem of concept drifting efficiently. Our work has proved to be capable of capturing emerging concepts in a manner of any time. Currently we are improving our approach in terms of the computational performance, and storage complexity by using a data-mining-ready data structure for a variety of streaming data. And we are also extending our algorithm to synthesize decision trees with the user-defined arbitrary time-ranges validation, compared with the current sliding window approach. For the users, it is essential for them to be able to specify starting time or end time of their decision tree according to the domain knowledge that only user knows. Moreover, our approach can be used for users to make a combination of criteria for different properties of decision tree e.g. the support examples or overall accuracy of the decision trees. In this way our algorithm can also be used as an experimental tool for the human interactive knowledge discovery.

References

1. Schlimmer, J.C., Richard II. Granger, J.: Beyond incremental processing: Tracking concept drift. In: AAAI National Conference on Artificial Intelligence, Philadelphia, PA, USA (1986) 502–507
2. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA (2001) 97–106
3. Jin, R., Agrawal, G.: Efficient decision tree construction on streaming data. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (2003)
4. Kalles, D., Morris, T.: Efficient incremental induction of decision trees. Machine Learning **24** (1996) 231–242
5. Wang, H., Fan, W., Yu, P., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA (2003)
6. Street, W.N., Kim, Y.: A streaming ensemble algorithm (sea) for large-scale classification. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA (2001) 377–382
7. Kolter, J.Z., Maloof, M.A.: Dynamic weighted majority: A new ensemble method for tracking concept drift. In: International Conference on Data Engineering, Bangalore, India (2003)
8. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, USA (1993)
9. Colomb, R.M.: Representation of propositional expert systems as partial functions. Artificial Intelligence **109** (1999) 187–209