International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 2 | ISSN : 2456-3307

Online Ensemble Learning of Data Streams with Gradually Evolved

A. Auysha[1], Dr. A. Jayachandran[2]

[1]PG Scholar, Department of M.Sc(Software Engineering), PSN College of Engineering & Technology, Tirunelveli, Tamilnadu, India
[2]Research Supervisor, Department of M.Sc(Software Engineering), PSN College of Engineering & Technology, Tirunelveli, Tamilnadu, India

ABSTRACT

Class evolution is an important research topic for data stream mining. All previous studies implicitly regard class evolution as a transient change, which is not true for many real-world problems. This paper concerns the scenario where classes emerge or disappear gradually. A class-based ensemble approach, namely class-based ensemble for class evolution. Emprical studies demonstrate the effectiveness of CBCE in various class evolution scenarios in comparison to existing class evolution adaptation methods. With the rapid development of incremental learning and online, learning ,mining tasks in the context of data stream have been widely studied. Generally data stream mining refers to the mining task that are conducted on a sequence of rapidly arriving data records. As the environment where the data are collected may change dynamically, the data distribution may also change accordingly. This phenomenon referred to as concept drift is one of the most important challenges in data stream mining. A data stream mining technique should be capable of constructing and dynamically updating a model in order to learn dynamic changes.

Keywords : Class-based ensemble for class evolution, class incremental learning, Drift detection method, Machine learning repository, Twitter crawl dataset and chunk-by-chunk.

I. INTRODUCTION

It is the process of extracting knowledge structures from continuous, rapid data records. A data stream is an ordered sequence of instances that in many applications of data stream mining can be read only once or a small number of times using limited computing and storage capabilities. Examples of data streams include computer network traffic, phone conversations, ATM transactions, web searches, and sensor data. Data stream mining can be considered a subfield of data mining, machine learning, and knowledge discovery.

In many data stream mining applications, the goal is to predict the class or value of new instances in the data stream given some knowledge about the class membership or values of previous instances in the data stream. Machine learning techniques can be used to learn this prediction task from labeled examples in an automated fashion. Often, concepts from the field of incremental learning, a generalization of Incremental heuristic search are applied to cope with structural changes, on-line learning and real-time demands. In many applications, especially operating within non-stationary environments, the distribution underlying the instances or the rules underlying their labeling may change over time, i.e. the goal of the prediction, the class to be predicted or the target value to be predicted, may change over time. This problem is referred to as concept drift.

To deal with these huge amounts of data in a responsible way, green computing is becoming a necessity. Green computing is the study and practice of using computing resources efficiently. A main approach to green computing is based on algorithmic efficiency. The amount of computer resources required for any given computing function depends on the efficiency of the algorithms used. As the cost of hardware has declined relative to the cost of Energy, the energy efficiency and environmental impact of computing systems and programs are receiving increased attention. More recently the need to process larger amounts of data has motivated the field of data mining. Ways are investigated to reduce the computation time and memory needed to process large but static data sets. In analogy to a database-management system, we can view a stream processor as a kind of data-management system, the high-level organization of which is suggested in. Any number of streams can enter the system. Each stream can provide elements at its own schedule; they need not have the same data rates or data types, and the time between elements of one stream need not be uniform. The fact that the rate of arrival of stream elements is not under the control of the system distinguishes stream processing from the processing of data that goes on within a database-management system. The latter system controls the rate at which data is read from the disk, and therefore never has to worry about data getting lost as it attempts to execute queries. Is a computer programming paradigm, equivalent to dataflow programming, event stream processing, and reactive programming that allows some applications to more easily exploit a limited form of parallel processing. Such applications can use multiple computational units, such as the FPUs on a GPU or field programmable gate arrays (FPGAs), without explicitly managing allocation, synchronization, or communication among those units.

Kernel functions are usually pipelined, and local on-chip memory is reused to minimize external memory bandwidth. Since the kernel and stream abstractions expose data dependencies, compiler tools can fully automate and optimize on-chip management tasks. Stream processing hardware can use core boarding, for example, to launch DMA sat runtime, when dependencies become known. The elimination of manual DMA management reduces software complexity, and the elimination of hardware caches reduces the amount of the area not dedicated to computational units such as ALUs.

From the core PrefDB query processing strategies that blend preference evaluation into query processing, we have also implemented a set of plug-in methods, which are described in the Appendix. Below is an overview of the core PrefDB modules

- ✓ The profile manager selects from the database preferences that can be combined with the conditions of the issued query. For this purpose, we use the preference selection algorithm proposed in [20]
- ✓ The query parser takes as input the query and preferences and generates an extended query plan that is passed to the PrefDB query optimizer.
- ✓ The query optimizer improves the input plan by applying a set of algebraic rules. This improved plan and a cost model for preference evaluation are used for generating alternative plans that interleave preference evaluation and query processing in different ways and for picking the plan with the cheapest estimated cost.
- ✓ The execution engine realizes the execution of the query plan selected by the query optimizer using one of our execution methods. We discuss

II. METHODS AND MATERIAL

A. Related Work

The concept of preference-aware query processing appears in many applications, where there is a matter of choice among alternatives, including query personalization [10], [18], [20], recommendations [4] and multi-criteria decision making [9], [13]. We discuss prior work with respect to how preferences are represented in the context of relational data and how they are integrated and processed in queries. In representing preferences, there are two approaches. In the qualitative approach, preferences are specified using binary predicates called preference relations [5], [10], [18]. In quantitative approaches, preferences are expressed as scores assigned to tuples [6], [23] be specified based on any combination of scores, confidences and context. Our framework allows us to process in a uniform way all these different query and preference types. In terms of preference integration and processing, one approach is to translate preferences into conventional queries and execute them over the DBMS [14], [19], [20], [21], [24]. Several efficient algorithms have been proposed for processing different types of queries, including top-k queries [13] and skylines [9]. These algorithms as well as query translation methods are typically implemented outside the DBMS. Thus, they can only apply coarse grained query optimizations, such as reducing the number of queries sent to the DBMS. Further, as we will also demonstrate experimentally plug-in methods do not scale well when faced with multi-join queries or queries involving many preferences. Native implementations modify the database engine by adding specific physical operators and algorithms. RankSQL [23] extends the relational algebra with a new operator called rank that enables pipelining and hence optimizing top-k queries. Another example of operator is the winnow operator [10], which selects all tuples corresponding to the Pareto optimal set. Our approach is different from existing works in several ways. First, existing techniques are limited to a particular type of query. In contrast to these approaches, we consider preference evaluation (how preferences are evaluated on data) and selection of the preferred tuples that will comprise the query answer as two operations. We focus on preference evaluation as a single operator that can be combined with other operators and we use its algebraic properties in order to develop generic query optimization and processing techniques. Finally, we follow a hybrid implementation that is closer to the database than plug-in approaches yet not purely native, thus combining the pros of both worlds. A different approach to flexible processing of queries with preferences is enabled in FlexPref [22]. FlexPref allows integrating different preference algorithms into the database with minimal changes in the database engine by simply defining rules that determine the most preferred tuples. Once these rules are specified a new operator can be used inside queries. It is worth noting that both FlexPref and our work are motivated by the limitations of plug-in and native approaches. FlexPref approaches the problem from an extensibility viewpoint. Our focus is on the problem of preference evaluation as an operator that is separate from the selection of preferred answers, and we study how this operator can be integrated into query processing in an effective yet not obtrusive to the database engine way.

B. Proposed Methodology

In this paper, we first construct an extended query plan that contains all operators that comprise a query and we optimize it. Then, for processing the optimized query plan, our general strategy is to blend query execution with preference evaluation and leverage the native query engine to process parts of the query that do not involve a prefer operator. Given a query with preferences, the goal of query optimization is to minimize the cost related with preference evaluation. Based on the algebraic properties of prefer, we apply a set of heuristic rules aiming to minimize the number of tuples that are given as input to the prefer operators. We further provide a cost-based query optimization approach. Using the output plan of the first step as a skeleton and a cost model for preference evaluation, the query optimizer calculates the costs of alternative plans that interleave preference evaluation and query processing in different ways. Two plan enumeration methods, i.e., a dynamic programming and a greedy algorithm are proposed. For executing an optimized query plan with preferences, we describe an improved version of our processing algorithm (GBU) (an earlier version is described in. The improved algorithm uses the native query engine in a more efficient way by better grouping operators together and by reducing the out-of-the-engine query processing.

Modules:

Registration & Interest Sum up
Query Formation
Query Optimization & Execution

A preferential query combines p-relations, extended relational and prefer operators and returns a set of tuples that satisfy the boolean query conditions along with their score and confidence values that have been calculated after evaluating all prefer operators on the corresponding relations. Intuitively, the better a tuple matches preferences and the more (or more confident) preferences it satisfies, the higher its final score and confidence will be, respectively. The query parser adds a prefer operator for each preference. Finally, the query parser checks for each preference, whether it involves an attribute (either in the conditional or the scoring part) that does not appear in the query and modifies project operators, such that these attributes will be projected as well of tuples that are given as input to the prefer operators. We further provide a cost-based query optimization approach into the database with minimal Proportional to the number of tuples flowing through the operators in the query plan. Assuming a fixed position for the other operators, the goal of our query optimizer is essentially to place the prefer operators inside the plan, such that the number of tuples flowing through the score tables is minimized. The execution engine of PrefDB is responsible for processing a preferential query and supports various algorithms. Existing techniques are limited to a particular type of query. In contrast to these approaches, we consider preference evaluation (how preferences are evaluated on data) and selection of the preferred tuples that will comprise the query answer as two operations.

III. RESULTS AND DISCUSSION

The implementation results can be shown as figure below

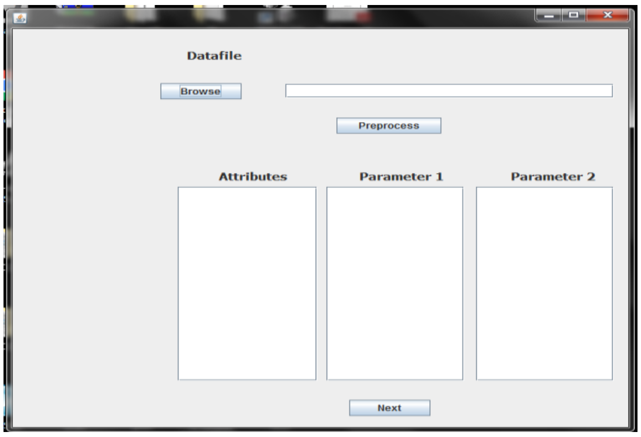


Figure 1: Upload Dataset

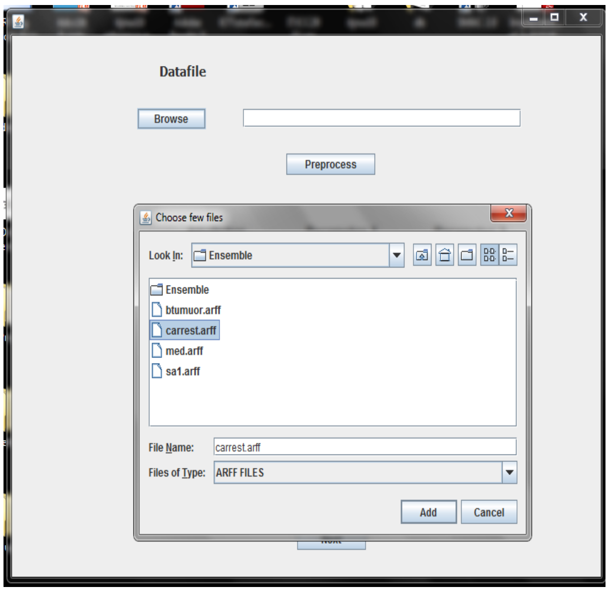
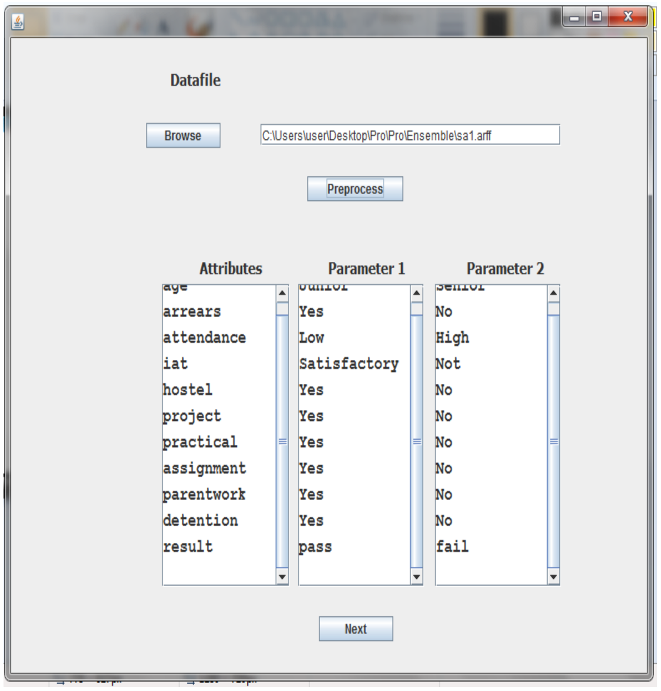
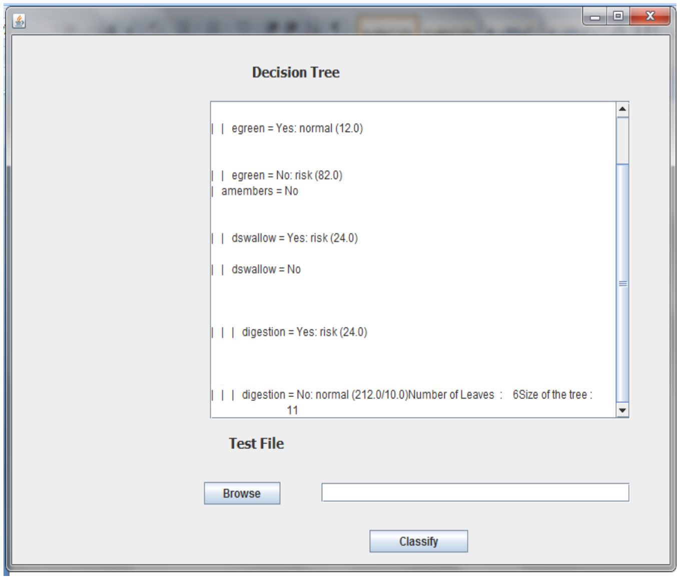


Figure 2 : Browse a Dataset



Attributes	Parameter 1	Parameter 2
arrears	Yes	No
attendance	Low	High
iat	Satisfactory	Not
hostel	Yes	No
project	Yes	No
practical	Yes	No
assignment	Yes	No
parentwork	Yes	No
detention	Yes	No
result	pass	fail

Figure 3: Pre processing



```
| |  egreen = Yes: normal (12.0)

| |  egreen = No: risk (82.0)
|  amembers = No

| |  dswallow = Yes: risk (24.0)

| |  dswallow = No

| | |  digestion = Yes: risk (24.0)

| | |  digestion = No: normal (212.0/10.0)Number of Leaves  :   6Size of the tree :
          11
```

Figure 4 : Decision tree

During Registration, each and every user will provide their basic information for authentication. After that, user has to provide their profile information and their interests about their movie. Based upon their, and with our movie datasets, we can be able to analyze their interest about the movie and have to provide the recommended movies to the particular user

IV.CONCLUSION

In this project, previous investigations on data stream mining assume class evolution to be the transient changes of classes, which does not hold for many real-world scenarios. In this work, class evolution is modelled as a gradual process, i.e., the sizes of classes increase or shrinks gradually. A new data stream mining approach, CBCE, is proposed to tackle the class evolution problem in this scenario. CBCE is developed based on the idea of a class-based ensemble. Specifically, CBCE maintains a base learner for each class and updates the base learners whenever a new example arrives. Furthermore, a novel under-sampling method is designed for handling the dynamic class-imbalance problem caused by gradually evolved classes.

V. REFERENCES

[1]. M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," SIGMOD Rec., vol. 34, no. 2, pp. 18–26, 2005.

[2]. P. Domingos and G. Hulten, "Mining high-speed data streams," in Proc. 6th ACM SIGKDD Int. Conf. Know. Discovery Data Mining, 2000, pp. 71–80.

[3]. J. Gama, I. Zliobait e, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," ACM Compute. Surv. vol. 46, no. 4, pp. 44:1–44:37, Mar. 2014.

[4]. L. Minku, A. White, and X. Yao, "The impact of diversity on online ensemble learning in the presence of concept drifts," IEEE Trans. Know. Data Eng., vol. 22, no. 5, pp. 730–742, May 2010.

[5]. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," IEEE Trans. Know. Data Eng., vol. 24, no. 4, pp. 619–633, Apr. 2012.

[6]. A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavalda, "New ensemble methods for evolving data streams," in Proc. 15th ACM SIGKDD Int. Conf. Know. Discovery Data Mining, 2009,pp. 139–148.

[7]. J. Liu, X. Li, and W. Zhong, "Ambiguous decision trees for mining concept-drifting data streams," Pattern Recog. Lett. vol. 30, no. 15,pp. 1347–1355, 2009.

[8]. Z.-H. Zhou and Z.-Q. Chen, "Hybrid decision tree," Know.-Based Syst., vol. 15, no. 8, pp. 515–528, 2002.

[9]. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham,"Integrating novel class detection with classification for concept drifting data streams," in Proc. Eur. Conf. Mach. Learn. Know. Discovery Databases, 2009, vol. 5782, pp. 79–94.

[10]. M. Masud, Q. Chen, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham, "Addressing concept-evolution in concept drifting data streams," in Proc. IEEE 10th Int. Conf. Data Mining, Dec. 2010, pp. 929–934.