A Data Stream Mining System

Hetal Thakkar Barzan Mozafari Carlo Zaniolo
University of California at Los Angeles
{hthakkar, barzan, zaniolo}@cs.ucla.edu

Abstract

On-line data stream mining has attracted much research interest, but systems that can be used as a workbench for online mining have not been researched, since they pose many difficult research challenges. The proposed system addresses these challenges by an architecture based on three main technical advances, (i) introduction of new constructs and synoptic data structures whereby complex KDD queries can be easily expressed and efficiently supported, (ii) an integrated library of mining algorithms that are fast & light enough to be effective on data streams, and (iii) support for Mining Model Definition Language (MMDL) that allows users to define new mining algorithms as a set of tasks and flows. Thus, the proposed system provides an extensible workbench for online mining, which is beyond the existing proposals for even static mining.

1 Introduction

On-line data stream mining plays a key role in growing number of real-world applications, including network traffic monitoring, intrusion detection, web click-stream analysis, and credit card fraud detection. Thus, many research projects have recently focused on designing fast mining algorithms, whereby massive data streams can be mined with real-time response [10, 4, 13, 7]. Similarly, many research projects have also focused on managing the data streams generated from these applications [9, 1, 6]. However, the problem of supporting mining algorithms in such systems has, so far, not received much research attention [12]. This situation seems unusual, since the need for a mining system for static data mining, was immediately recognized [8] and has lead to systems such as, Weka [5] and OLE DB for DM [11]. Furthermore, static mining algorithms can also be written in procedural language using a cache mining approach that makes little use of DBMS essentials. However, online mining tasks cannot be deployed as stand-alone algorithms, since they require many DSMS essentials, such as I/O buffering, windows, synopses, load shedding, etc. Clearly, KDD researchers and practitioners would rather concentrate on the complexities of data mining tasks and avoid the complexities of managing data streams, by letting the mining system handle them. In short, while mining systems are a matter of convenience for stored data, they are a matter of critical necessity for data streams. Thus, this demo presents the SMM system, namely Stream Mill Miner, which is specifically designed to address this critical necessity. Building such a system raises difficult research issues, which SMM solves through an architecture based on three main technical advances, as follows.

- Extending recently developed DSMSs, which are currently designed to only support simple queries, to express complex mining queries,
- Integrating a library of mining algorithms that are fast & light enough to be effective on data streams, and
- Supporting a higher level mining language, namely Mining Model Definition Language (MMDL), which allows definition of mining models that encapsulate related mining tasks and mining flows for ease-of-use and extensibility.

Thus, SMM extends an existing DSMS, namely Stream Mill, with user-friendly, high-level mining models that are implemented with a powerful SQL-based continuous query language, namely Expressive Stream Language (ESL). ESL is an extension of SQL based on User Defined Aggregates (UDAs). Therefore, this demo presents the following key features and methods of SMM.

- Mining models and their use for online classification, clustering, and association rule mining,
- Generic support for advanced meta concepts to improve accuracy of classifiers, e.g. ensembles, and
- Definition of mining algorithms consisting of multiple processing steps as mining flows in MMDL.

2 High-Level Mining Models

An on-line data stream mining system should allow the user to (i) define new mining models and (ii) uniformly invoke diverse set of built-in and user-defined mining algorithms. Existing solutions for static data mining, do not allow (i) and simply focus on (ii), which results in a close system. However, online mining systems must provide an open framework, since new on-line mining algorithms are constantly being proposed [10]. SMM achieves both of these goals via supporting MMDL as we discuss next.

SMM allows the user to define new mining models by specifying the tasks that are associated with the model. For instance, most classifiers will consist of two tasks, learning and predicting, whereas association rule mining consists of finding frequent patterns and deriving rules from them, and so on. Furthermore, data cleaning and post-analysis steps can also be specified as tasks. Finally, the analyst

can specify mining flows that connect these tasks to implement complex mining process, such as ensemble based methods [13, 4, 7]. The model definition specifies the tables that are shared by different tasks of the model. Thus, different instances of the model will work on separate instances of these tables, but the tasks of the same model instance share these tables. Additionally, the model definition associates a UDA with each individual task of the model as discussed in Section 3.

Example 1 defines a simple Naive Bayesian Classifier (NBC) and creates an instance of this model type in MMDL. In Example 1, the UDAs associated with the **Learn** and **Classify** tasks are **LearnNaiveBayesian** (omitted due to space constraints) and **ClassifyNaiveBayesian** (Example 3), respectively. Thus, MMDL allows the users to create arbitrary mining models and instantiate them uniformly. Once a mining model instance is created, the user can then invoke different tasks of the model with a consistent syntax. For instance, Example 2 invokes the **Learn** task of the NBC instance created in Example 1. Note, we omit the discussion of the formal syntax here, due to space constraints.

Example 1 *Defining A ModelType for an NBC*

```
CREATE MODEL TYPE NaiveBayesianClassifier {
    SHAREDTABLES (DescriptorTbl),
    Learn (UDA LearnNaiveBayesian,
        WINDOW TRUE, PARTABLES(),
        PARAMETERS()
      ),
    Classify (UDA ClassifyNaiveBayesian,
        WINDOW TRUE, PARTABLES(),
        PARAMETERS()
      )
};
CREATE MODEL INSTANCE NaiveBayesianInstance
        AS NaiveBayesianClassifier;
```

Example 2 *Invoking the Learn task of the NBC Instance*

```
RUN NaiveBayesianInstance.Learn WITH TrainingSet;
```

In Example 2, the **TrainingSet** is assumed to have the same schema as expected by the UDA associated with the **Learn** task–the system checks this automatically. Furthermore, the RUN statement allows an additional USING clause to specify the parameters required by the mining task. The USING clause is omitted in Example 2, since there are no additional parameters. However, advanced mining algorithms can be customized with the USING clause as seen in Section 4. Next, we discuss *generic* implementation of UDAs that are associated with these mining tasks.

3 On-line Mining in SMM

We use Naïve Bayesian Classifier (NBC) as an example to explain *generic* implementation of on-line mining algorithms in SMM. Before we move to data streams, let us first consider the situation where we want to learn an NBC over a static table, such as that of Table 1.

RID	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	Yes
3	Overcast	Hot	High	Weak	Yes
...	...	...	...	..	...

Table 1. The relation PlayTennis

RID	Column	Value	Dec
1	1	Sunny	No
1	2	Hot	No
1	3	High	No
1	4	Weak	No
2	1	Sunny	Yes
2	2	Hot	Yes
2	3	High	Yes
2	4	Strong	Yes
...	...	...	...

Table 2. Verticalized PlayTennis Relation

Building an NBC from a table like this (containing the training tuples), only requires counting (i) the total number of tuples, (ii) the number of tuples belonging to each class, and (iii) the number of tuples belonging to each class for each (column, value) pair. This is also known as the descriptive phase. These counts can be obtained with a set of SQL queries, one for each attribute, and they are stored in a table, whereby the probability of the 'Yes' and 'No' decision for a new tuple, is then obtained via simple mathematical computations. However, repeating the computation of counts for each attribute is inefficient and requires writing different queries for tables having different number of columns. This problem is addressed by assuring *genericity* in SMM.

3.1 Genericity

Genericity is an important property provided by systems such as Weka and OLE DB for DM, which allow their mining algorithms to be applied over arbitrary tables [12]. Thus, SMM also supports *genericity* via an approach similar to Weka, namely *verticalization*. SMM *verticalizes* input tuples into column/value pairs, e.g., the first two tuples in Table 1 are represented by the eight tuples shown in Table 2. This *verticalization* is realized in SMM by user-defined table functions. Therefore, tuples of any arbitrary schema are converted into this vertical format and processed through UDAs, which specify arbitrarily complex tasks. Thus, we next discuss the support for these UDAs.

3.2 UDAs

Let's assume that instead of a static table, we want to classify a stream of tuples. In this case a continuous query would be required to classify the incoming tuples. However, simple continuous queries are not enough to express many complex mining tasks. Thus, SMM supports UDAs and windows/slides over UDAs, to express these complex

queries efficiently. Therefore, the **ClassifyNaiveBayesian** UDA, given in Example 3, classifies test tuples based on the **DescriptorTbl**, built in the descriptive phase.

Example 3 *NB Classification Aggregate*

```
DescriptorTbl(Col INT, Val INT, Dec INT, Count REAL)
AGGREGATE ClassifyNaiveBayesian(col INT,
          val CHAR(10)): CHAR(3) {
    TABLE pred(dec INT, tot REAL);
    INITIALIZE: {
      INSERT INTO pred SELECT Dec, abs(log(Count+1))
                          FROM DescriptorTbl;
    } ITERATE: {
      UPDATE pred p SET tot = tot +
          (SELECT abs(log(Count+1))
          FROM DescriptorTbl WHERE Val = val
            AND Col = col AND Dec = p.dec);
    } TERMINATE: {
      INSERT INTO RETURN
        SELECT p.dec FROM pred p
        WHERE NOT EXIST (
          SELECT * FROM pred p1
          WHERE p1.tot > p.tot
            OR (p1.tot = p.tot AND p1.dec < p.dec));
    }
}
```

The **ClassifyNaiveBayesian** UDA (Example 3) sums up the probabilities of each outcome, up on arrival of each vertical tuple in the INITIALIZE and ITERATE states. These states essentially maintain a sum for each **value-dec** combination. The TERMINATE state determines the most likely outcome based on the maintained *sums*. We note that the computation presented in **ClassifyNaiveBayesian** UDA is blocking, since it only returns the outcome up on seeing the end of the stream in the TERMINATE state. Therefore, this UDA cannot be applied directly over data streams. Therefore, SMM extends standard SQL:2003 windows over UDAs, as opposed to just built-in aggregates, to convert the blocking UDAs to non-blocking ones. Thus, the **ClassifyNaiveBayesian** UDA is invoked as follows.

```
SELECT ClassifyNaiveBayesian(Col, Val)
    OVER (ROWS 4 PRECEDING SLIDE 5)
FROM VerticalStream
```

The ROWS 4 PRECEDING clause represents a window of 5 tuples (including the current tuple). Furthermore, the SLIDE 5 clause instructs SMM to return results every 5 tuples. In general, such windows allow invocation of blocking UDAs over streams, since the execution is constrained over the specified window. The queries like the one above are called tumbling window queries, since the slide size is greater than or equal to the window size [2]. In such cases, SMM repeats the following computation over each tumbling window, uniformly across arbitrary UDAs. SMM executes INITIALIZE for the first tuple, ITERATE for the next (window size - 1) tuples and then TERMINATE, since we reached the end of the window. This processing is repeated from the next (slide size - window size) tuples. Thus, in this case, after INITIALIZE, ITERATE is executed 4 times followed by TERMINATE. The processing is then repeated from the next tuple since slide size = window size. Therefore, UDAs along with windows and slides naturally integrate online prediction.

Now, we consider the case where the training set is also streaming. A similar solution, based on UDAs, can be applied in this case, which will enable the classifier to continuously learn new concepts that appear in the training stream. However, the classifier should also 'forget' the old concepts, which may reduce the accuracy of the classifier. Therefore, a windowed approach is more suitable, i.e. a classifier should be differentially maintained over a window of the training stream. Indeed, in data streams, many functions similarly need to be computed differentially, i.e. without recomputing everything upon arrival/expiration of a tuple. Therefore, SMM allows the users to define windowed version of their UDAs, where differential computation is declaratively specified via an additional EXPIRE state, which is invoked once for each tuple expiring out of the window. Due to space constraints we omit the detailed discussion of the windowed UDA that differentially maintains the statistics for an NBC, using the EXPIRE clause.

Indeed, UDAs along with different kinds of windows over them provide a great source of power and flexibility to SMM. Therefore, the user can define arbitrarily complex mining models and support them with UDAs.

4 New algorithms and methods

While many on-line mining algorithms are integrated in SMM, we only discuss Association Rule Mining (ARM) and ensemble based methods here, due to space constraints.

4.1 Association Rule Mining (ARM)

The first major task for ARM is the identification of frequent patterns, which has been the focus of many research efforts. For instance, the SWIM algorithm [10] differentially maintains frequent patterns over a large sliding window. SWIM uses FP-tree data structure to compute/store frequent patterns, whose frequencies are monitored with a fast verifier (i.e., an algorithm based on conditional counting, see [10] for definition of *verification*). An on-line mining system must integrate such new and advanced mining algorithms. For instance, Calders et al. [3], proposed an approach to incorporate ARM in relational databases through virtual mining views. This approach proposed that a mining system should provide a set of mining views for ARM and that the system executes appropriate mining methods when these views are queried.

SMM adopts a similar approach that defines a built-in mining model for ARM, as shown in Example 4. The model is again composed of two tasks, one for frequent patterns mining and another for determining association rules from the set of frequent patterns. Therefore, these tasks represent a mining flow, which can also be defined in the mining

model, e.g. **ARMFlow** in Example 4. Each mining flow has an input stream and an output stream, INSTREAM and OUTSTREAM, respectively. The analyst specifies how the tasks of a mining model interconnect via intermediate streams, e.g. **FrequentPatterns** stream. Other tasks, such as cleaning and post-analysis, can also be added to such flows. Thus, naive users do not require in-depth knowledge of each task and can simply invoke a mining flow that represents the complete mining process.

Example 4 *ModelType for Association Rule Mining*

```
CREATE MODEL TYPE AssociationRuleMiner {
    SHAREDTABLES (Sets),
    FrequentItemsets (UDA FindFrequentItemsets,
        WINDOW TRUE, PARTABLES(FreqParams),
        PARAMETERS(support Int)
      ),
    AssociationRule (UDA FindAssociationRules,
        WINDOW TRUE, PARTABLES(AssocParams),
        PARAMETERS(confidence Real)
      ),
    Flow ARMFlow (
      CREATE STREAM FrequentPatterns AS
        RUN FrequentItemsets ON INSTREAM;
      INSERT INTO OUTSTREAM RUN AssociationRules
        ON FrequentPatterns USING confidence > 0.60;
      )
};
CREATE MODEL INSTANCE AssociationRuleMinerInst
          AS AssociationRuleMiner;
```

In this case, the UDAs associated with these tasks are implemented externally in C/C++, which enables integration of advanced algorithms, such as SWIM. The overhead of this integration is negligible based on our experiments.

4.2 Ensemble Based Methods

A core issue in online data mining is concept drift/shifts, which are caused by the changes in the data distribution or underlying concept. The problem has been studied in detail by [13, 4, 7] and ensemble based methods are proposed as an effective solution. These techniques essentially learn an ensemble of classifiers and use their accuracy on the training stream to improve the accuracy of final classification over the testing stream. Thus, it is imperative that such techniques are integrated in an on-line mining system. SMM generically supports such techniques, i.e. these techniques can be applied over any arbitrary classifier such as NBC, decision tree classifier, etc. Analysts can essentially define mining flows similar to the one defined in Example 4.

5 SMM Architecture

SMM employs a client-server architecture, where multiple clients may be connected to a single server. The client simply sends user commands to the server. The sever consists of 3 main components, I/O scheduler (IOS), Query Scheduler (QS), and the compiler. The IOS communicates with the clients and the data sources. The QS is responsible for concurrently executing the queries. Finally, the compiler compiles user defined entities, such as UDAs, queries, run task statements, etc., to C/C++ code that is executable.

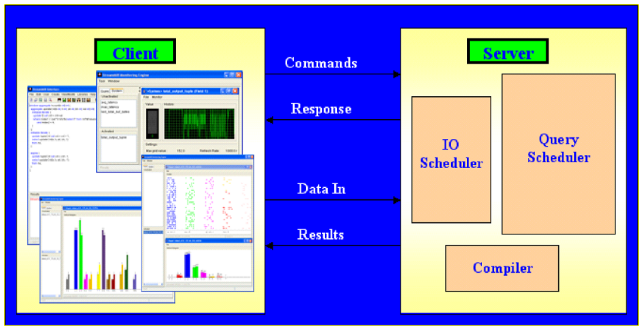

Figure 1. SMM Architecture

6 Conclusion

In this demo, we will show a data stream mining system that goes beyond the existing solutions for even static mining, by providing an extensible mining workbench. We will also demonstrate integration of user defined mining algorithms, including advanced algorithms, such as SWIM, ensemble based methods, etc., in SMM using MMDL.

References

[1] A. Arasu, S. Babu, and J. Widom. CQL: A language for continuous queries over streams and relations. In *DBPL*, 2003.

[2] Y. Bai, H. Thakkar, C. Luo, H. Wang, and C. Zaniolo. A data stream language and system designed for power and extensibility. In *CIKM*, 2006.

[3] T. Calders, B. Goethals, and A. Prado. Integrating pattern mining in relational databases. In *PKDD*, 2006.

[4] F. Chu and C. Zaniolo. Fast and light boosting for adaptive mining of data streams. In *PAKDD*, volume 3056, 2004.

[5] Weka 3: data mining with open source machine learning software in java. http://www.cs.waikato.ac.nz.

[6] D. Abadi et al. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 2003.

[7] George Forman. Tackling concept drift by temporal inductive transfer. In *SIGIR*, pages 252–259, 2006.

[8] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Commun. ACM*, 1996.

[9] Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. Data models and query language for data streams. In *VLDB*, 2004.

[10] B. Mozafari, H. Thakkar, and C. Zaniolo. Verifying and mining frequent patterns from large windows over data streams. In *ICDE*, 2008.

[11] Z. Tang and et al. Building data mining solutions with OLE DB for DM and XML analysis. *SIGMOD Record*, 2005.

[12] H. Thakkar, B. Mozafari, and C. Zaniolo. Designing an inductive data stream management system: the stream mill experiences. In *Scalable Stream Processing Systems*, 2008.

[13] H. Wang and et al. Mining concept-drifting data streams using ensemble classifiers. In *SIGKDD*, 2003.