StreamDM: Advanced Data Mining in Spark Streaming

Albert Bifet
Télécom ParisTech
Paris, France
bifet@telecom-paristech.fr

Silviu Maniu
Université Paris-Sud
Orsay, France
maniu@lri.fr

Jianfeng Qian Guangjian Tian Cheng He
Huawei Noah's Ark Lab
Hong Kong SAR, China
jianfeng.qian@outlook.com
{tian.guangjian,hecheng}@huawei.com

Wei Fan
Baidu Big Data Lab
Sunnyvale, CA, US
wei.fan@gmail.com

***Abstract*—Real-time analytics are becoming increasingly important due to the large amount of data that is being created continuously. Drawing from our experiences at Huawei Noah's Ark Lab, we present and demonstrate here StreamDM, a new open source data mining and machine learning library, designed on top of Spark Streaming, an extension of the core Spark API that enables scalable stream processing of data streams. StreamDM is designed to be easily extended and used, either practitioners, developers, or researchers, and is the first library to contain advanced stream mining algorithms for Spark Streaming.**

***Keywords*-stream mining; Spark Streaming; open-source; software**

I. Introduction

Data originating from high-speed steams is more and more prevalent in today's data ecosystem. Examples of data streams are ones containing credit card transactions, Internet traffic data, sensor data, or network alarm data. In order that network operators are able to extract knowledge, find defects, and act on information present – either explicitly or implicitly – in the stream, the data needs to be analyzed and processed. This can pose a significant challenge in streams in which data arrives at high speed, and efficient data mining and machine learning algorithms are needed for that. Such algorithms need hence to be extremely time-efficient while using small amounts of memory.

To deal with the increasing quantity of data, various platforms for parallel data processing have been proposed. The most used platform is MapReduce [1], with its open-source implementation, Hadoop[1]. MapReduce allows full parallelization of data analysis task on clusters of commodity hardware. Lately, Spark [2][2] and its streaming version, Spark Streaming [3], built on top of Hadoop, have emerged as the go-to platforms for data analysis on clusters. Spark's advantages are that it facilitates better fault tolerance, in addition to faster data processing by using main-memory storage.

In addition, several open-source projects implementing methods for data streams are proposed, and we remind here the MOA [4] and SAMOA [5] Java-based libraries. The Spark packages MLlib and spark.ml provide machine learning algorithms, primarily for batch mode processing. Implemented in C++, Vowpal Wabbit[3] provides fast implementations of gradient descent algorithms.

Our StreamDM library aims to bridge the gap between the Spark Streaming environment and the open-source online data mining libraries, by providing an environment for data analysis on clusters. Its aim is to be complementary to the current methods available in MLlib and spark.ml, by focusing on advanced machine learning and data mining methods. In the medium-to-long term, the library is intended to be the gathering point of practical implementations and deployments for large-scale data streams.

StreamDM is designed to bring several improvements over the current machine learning Spark packages:

- **Ease of use** Experiments can be executed from the command-line, without the need for re-compiling the library for developing new mining tasks.
- **No dependence on third-party libraries** For example, MLlib uses the linear algebra package Breeze, which in turn depends on netlib-java (among others). Due to licensing issues, *netlib-java*'s native libraries are not included in MLlib's dependency set under default settings, complicating the compilation process. We aim to avoid this in StreamDM.
- **Ease of extensibility**, relying on a simple, but powerful and extensible class hierarchy.
- **Advanced stream machine learning / data mining methods**: streaming decision trees,, streaming clustering methods such as CluStream and StreamKM++. To the best of our knowledge, this is the first implementation of advanced methods on top of Spark Streaming.

[1]http://hadoop.apache.org
[2]http://spark.apache.org
[3]https://github.com/JohnLangford/vowpal_wabbit

StreamDM is open-source software and is available at https://github.com/huawei-noah/streamDM. It is implemented in Scala, under the Apache Software License v2.0. Examples for getting started, programming guide, and the full API documentation can be found at http://huawei-noah.github.io/streamDM/.

II. StreamDM Description

StreamDM is designed on top of Spark Streaming, and benefits from being inside the Hadoop open-source ecosystem. Its Scala source code allows for functional programming, and minimizes chances of side-effects in the code. Scala is a functional language which runs on top of the Java VM, and it is fully compatible with any Java-based library.

StreamDM is designed with two types of user in mind:

- *practical application users*, who can use the available implemented algorithms directly, by defining task and using command-line parameters without needing to explicit develop them, and
- *developers and researchers*, who can easily develop new algorithms on top of StreamDM.

A. Design

StreamDM uses Spark Streaming as the provider of streaming data. In Spark Streaming, datasets are divided in several *discretized streams* (DStream) which are then processed continuously. Note that this means that Spark Streaming is not a one-by-one streaming environment; instead, the data is processed in streaming mini-batches. Its main disadvantage is that the latency of data becomes of the order of seconds. However, the main advantage is that it allows combining batch processing algorithms with streaming algorithms.

The main execution block in StreamDM is the Task, which defines streaming data mining workflows. This allows users to program complex tasks, even containing multiple layers of training and evaluation. Tasks are designed to be independent from the algorithms that are used; they are defined only in terms of high-level class types, i.e., classes derived for the base Scala traits shown in Table I.

Each of the above class types gets a stream as an input, and outputs also a stream. Internally, the streams are represented as Spark Streaming DStream, containing our internal instance data structure, the Example. An Example encompasses input and output Instance and a weight; in turn, the Instance can contain data structure depending of the input format of the streams (e.g., dense instances in CSV text format, sparse instances in LibSVM format, and text instances). All operations are made on the Example; this allows for task design without the need to know the underlying implementation of the instances. It also allows streams which may change their underlying format at any time.

In general, the task process proceeds as follows. First, the streaming data from DStream is read and parsed by a Reader class. Then, the training and testing streams are passed into a Learner (which contains the data mining / machine learning algorithm implementation). The assignments or predictions from Learner are evaluated by an Evaluator – e.g., a class transforming the predicted instances into a confusion matrix. Finally, the results are output by a Writer class – to the console, disk or HDFS files, or as streams for other tasks.

In terms of processing work, the main components for creating learning tasks are learners. Learners can be added by implementing the Learner trait and its associated methods: .init for initializing the Model inside the learner, and .train for updating the model with the data from the stream. The Model trait allows data structures to exist inside learners, e.g., a linear model in SGD, or micro-clusters in clustering. For more specific uses, specialized traits need to be implemented instead. For example, the Classifier trait also contains a .predict methods which applies the model to a stream, and predicts the label of an input instance.

For illustration, Figure 1 presents the flow of an predict-then-train streaming task, EvaluatePrequential. First, a StreamReader parses the input stream and creates a DStream of Example, which then is used for training in a Classifier, which updates its underlying Model. In parallel, the input stream is used for predicting the labels, generating tuples of the original Example and a Double representing the predicted label. This tuple is then passed through an Evaluator, which generates an output stream of Strings to be written by a StreamWriter.

A seen above, EvaluatePrequential can be used with any classifier, by just using command line options. An example command-line using logistic Stochastic Gradient Descent is the following:

```
./execute_task.sh "EvaluatePrequential -l   (
    SGDLearner -l 0.01 -o LogisticLoss -r
    ZeroRegularizer)"
```

Note that EvaluatePrequential is not the only task possible in the StreamDM framework. It is illustrated here to serve as an example on how tasks can be easily implemented.

B. Implemented Learner Algorithms

StreamDM contains implementations of the following classification algorithms, which implement the Classifier trait:

- **Multinomial Naive Bayes [6]:** Learner that models a document as a bag-of-words. For each class, the probability of observing a word given that class

Table I
TASK BUILDING BLOCKS.

trait	objective
StreamReader	read and parse Example and create a stream
Learner	provides the train method from an input stream
Model	data structure and set of methods used for Learner
Evaluator	evaluation of predictions
StreamWriter	output of streams

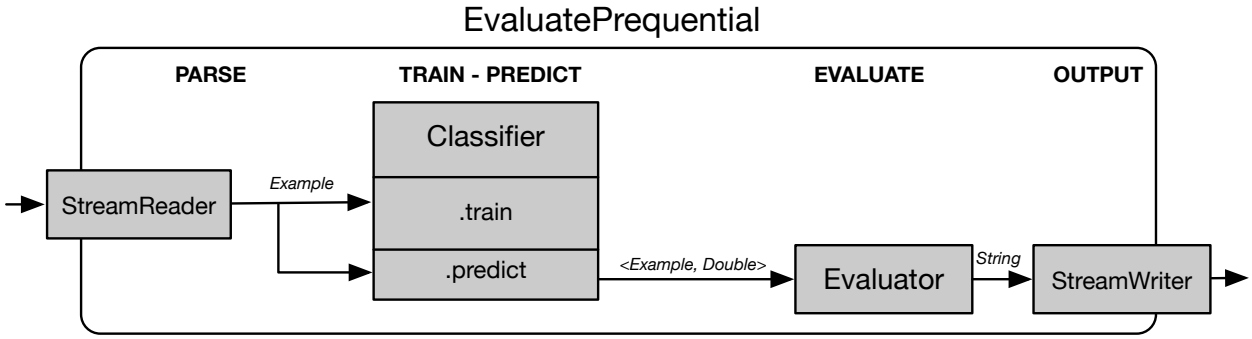

Figure 1. An example of a STREAMDM task, EvaluatePrequential.

is estimated from the training data, simply by computing the relative frequency of each word in the collection of training documents, for that class. The classifier also requires the prior probabilities, which are straightforward to estimate from the frequency of classes in the training set.

- **SGD Learner and Perceptron** Classifier that uses the Stochastic Gradient Descent optimizer for learning various linear models: binary class SVM, binary class logistic regression, and linear regression. This is achieved via the implementation of different loss functions (linear, logistic, hinge, perceptron) and different regularizers (L1, L2).
- **Hoeffding Decision Trees [7]:** Incremental decision tree learner for large data streams, that assumes that the data distribution is not changing over time. It grows incrementally a decision tree based on the theoretical guarantees of the Hoeffding bound (or additive Chernoff bound). A node is expanded as soon as there is sufficient statistical evidence that an optimal splitting feature exists, a decision based on the distribution-independent Hoeffding bound. The model learned by the Hoeffding tree is asymptotically nearly identical to the one built by a non-incremental learner, if the number of training instances is large enough.
- **Bagging [8]:** Ensemble method that improves the accuracy of a single classifier. Non-streaming bagging builds a set of base models, training each model with a bootstrap sample created by drawing random samples with replacement from the original training set. Each base model's training set contains each of the original training example, each weighted by a number following a binomial distribution. This binomial distribution tends towards a Poisson(1) distribution, for large values. Thus, the online version of the Bagging classifier, instead of using sampling with replacement, gives each example a weight according to a Poisson(1) distribution.

Also STREAMDM contains implementations of the following clustering methods, which implement the Clusterer trait:

- **CluStream [9]:** Clusterer based on the concept of microclusters, data structures which summarize a set of instances from the stream, and is composed of a set of statistics which are easily updated and allow fast analysis.
  CluStream has two phases. In the online phase, a set of microclusters are kept in main memory; each instance coming from the input stream can then be either appended to an existing microcluster or created as a new microcluster. Space for the new microcluster is created either by deleting a microcluster (by analyzing its expiration timestamp) or by merging the two closest microclusters. The offline phase will apply a weighted k-means algorithm on the microclusters, to obtain the final clusters from the stream.

- **Stream KM++ [10]:** Clusterer that computes a small weighted sample of the data stream,called the coreset of the data stream. A new data structure called coreset tree is developed in order to significantly speed up the time necessary for sampling non-uniformly during the coreset construction. After the coreset is extracted from the data stream, a weighted k-means algorithm is applied on the coreset to get the final clusters for the original stream data.

III. Demonstration Plan

In the demonstration, we will showcase our current library both from the point of view of users and from the point of view of developers. We plan to do achieve this in an interactive way with the atendees, and to focus on the practical applications of StreamDM. Another focus is to outline the interesting research challenges that are introduced in the context of a data stream library developed on Spark.

For *practitioners*, we plan to show:

- a demonstration of different data processing scenarios (classification, clustering and complex processing) having different sources of data;
- the use of various algorithms: how to use the task options in the command line, and how to use different evaluators and output options;
- the use of various sources of data such as network socket streams and disk files;
- benchmarks on the efficiency of StreamDM compared to other Spark implementations such as MLlib;
- accuracy and effectivenes benchmarks compared to other mature stream mining libraries, such as MOA and SAMOA.

By focusing on practical real-world scenarios, we wish to demonstrate that StreamDM is intuitive and that it is ready for industry deployments.

Another objective of StreamDM is to gather implementations of state-of-the-art algoritms studied in the data mining community in an open-source library. To this end, we wish to also target more technical developers, but also researchers wishing to deploy streaming machine learning models to a distributed cluster on Spark using StreamDM.

Hence, for *developers* and *researchers*, we plan to show:

- how new algorithms can be added by extending Learner and its specialized traits, and to show that they can be directly plugged into already implemented tasks;
- how new tasks can be developed for more complex research scenarios.

We would also like to discuss with experts and practitioners to get feedback and brainstorm research perspectives for further extension of our library. Particularly, we plan to focus on insights into the workings and drawbacks of different approaches. For example, we wish to show in which situation does an algorithm or an evaluation measure work or fail, in addition to research issues emerging from the hybrid streaming-batch processing model of Spark Streaming. We believe the audience will benefit from the demonstration of StreamDM and its potential for practical applications and data mining research.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 107–113.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012.

[3] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters," in *HotCloud*, 2012.

[4] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[5] G. De Francisci Morales and A. Bifet, "SAMOA: Scalable advanced massive online analysis," *Journal of Machine Learning Research*, vol. 16, pp. 149–153, 2014.

[6] A. Mccallum and K. Nigam, "A comparison of event models for naive bayes text classification," in *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[7] P. Domingos and G. Hulten, "Mining high-speed data streams," in *KDD '00: 6th international conference on Knowledge Discovery and Data mining*, 2000, pp. 71–80.

[8] N. Oza and S. Russell, "Online bagging and boosting," in *Artificial Intelligence and Statistics*, 2001, pp. 105–112.

[9] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *VLDB*, 2003, pp. 81–92.

[10] M. R. Ackermann, C. Lammersen, M. Märtens, C. Raupach, C. Sohler, and K. Swierkot, "StreamKM++: A clustering algorithm for data streams," *ACM Journal of Experimental Algorithmics*, vol. 17, 2012.