Technical Report 77

MOSIS: An Open Source Framework for Signal Processing and Machine Learning

Claas Ahlrichs
Michael Lawo

TZI, Universität Bremen

TZI-Bericht Nr. 77
2015

TZI-Berichte

Herausgeber:
Technologie-Zentrum Informatik und Informationstechnik
Universität Bremen
Am Fallturm 1
28359 Bremen
Telefon: +49 421 218 94090
Fax: +49 421 218 94095
E-Mail: hq@tzi.de
http://www.tzi.de

ISSN 1613-3773

MOSIS: An Open Source Framework for Signal Processing and Machine Learning

Ahlrichs, Claas
claasahl@tzi.de

Lawo, Michael
mlawo@tzi.de

May 27, 2015

Abstract

This paper introduces a modularized signal processing and analysis (MOSIS) framework for signal processing, stream analysis, machine learning and stream mining applications. Here the requirements and the design concept of the open source software framework are outlined. MOSIS is designed to ease benchmarking efforts and rapid prototyping of stream-based algorithms. Furthermore, it facilitates a design pattern driven approach and can easily be expanded to include new modules as well as algorithms. The nature of MOSIS enables use by researchers, enthusiasts and practitioners alike. At the end, the structure, documentation and code quality make it suitable for training and classroom scenarios. Here we motivate the framework and outline its status quo.

Keywords

Signal Processing, Machine Learning, Stream Mining, Signal Analysis, Design Patterns, Framework

1 Introduction And Motivation

Today's world generates enormous amounts of streaming data. With increased digitization and computerization vast data sources are present in almost every sector of modern living. Pretty much anything from sensors applications, social media streams, log files, surveillance, network and traffic monitoring to blog posts and emails can serve as a source of streaming data.

Time and space constraints play an important role and might not always allow for processing of data samples multiple times. Imagine a network router that analyzes network traffic in order to detect intrusion patterns. Such patterns need to be recognized as fast as possible and only a (very) limited number of network packets can be buffered (if any at all). Thus timing and memory requirements cannot be ignored, algorithms need to act within a limited window of time and space. Latter point might be more obvious, when crawling the Internet it is obvious that not all contents can remain in main memory and massive parallelism must be possible.

Typical tasks of stream-based algorithms include but are not limited to clustering, classification, outlier detection, trend analysis and querying (i.e. range queries, top-k queries). Applications of (evolutionary) stream analysis can yield to knowledge on "hot" topics on social media platforms or "hot" news stories. Other applications may focus on analysis of financial streaming data (i.e. currency exchange pairs, bonds, etc.) in order to generate trade signals. Applications also include the detection of abnormal heart rates or to detect symptoms in general (i.e. tremor [15, 16], dyskinesia [6, 18]).

Stream mining algorithms provide a trade-off between real time updates and memory usage at the cost of accuracy. They can be utilized to return answers with high probabilities of being accurate. In contrast to traditional mining algorithms, these algorithms require less memory (i.e. KB vs. GB) as much of their work is done in hash tables. Thus only a small portion of the data samples (if any at all) are maintained in memory. Anytime, the model of such an algorithm is queried then the answer is simply looked-up or computed. Thus real-time responses are possible.

A real world data stream or snapshot of such can present an arbitrary number of samples (e.g. several thousands, several millions or even an infinite number of samples) to an algorithm. Thus such algorithms have to operate under tight constraints in time and space which may be radically different from traditional batch processing settings. The most important ones can be summarized as follows [2]: (1) inspect each sample at most once, (2) work in a limited window of time and memory and (3) be ready to predict at any time. These are the core concepts of stream based learning algorithms. The algorithm's prediction model is continuously updated as new (and relevant) data samples pass through. It maintains such a model in order to be ready to predict at any time.

In comparison to traditional batch learning settings, stream learning is a "relatively" new field. Established frameworks like Waikato Environment for Knowledge Analysis (WEKA) are not available for stream learning or they have not reached such popularity within the community. Many researchers

still start from scratch when developing new algorithms. Furthermore evaluation practices are not as well defined in this area. The field has grown to include various stream learning algorithms, thus for a performance estimation it is crucial to provide a critical evaluation with competitive state-of-the-art solutions. The capacity to handle large (and possibly infinite) streams should be demonstrated as well as their space and time constraints. However, not all publications provide such information, thus making effective judgments on their applicability difficult and comparing them to newly developed algorithms tedious. This is where MOSIS comes in. It is intended to provide the community with:

- Collection of stream based algorithms as well as preprocessing and post-processing methods;
- A modularized and flexible way to manage information flow and ease rapid prototyping;
- Easy extensibility to add new modules (i.e. algorithms, filters, data sources, data sinks, etc.);
- A way to create and manage benchmarks for an easy comparison of algorithms and reproducibility.

Here researchers can simply built upon readily available filters, preprocessing modules and algorithms to speed-up their benchmarking and development efforts. These modules can be wired among each other to represent the flow of information. We show an example in Section 4 and Figure 1.

The remainder of this paper is organized as follows: Section 2 highlights related work to MOSIS, Section 3 and Section 4 list requirements and design aspects of MOSIS (respectively), Section 5 discusses a set of repositories employed by MOSIS and the framework's status quo is outlined in Section 6. Section 7 concludes this paper.

2 Related Work

WEKA is a machine learning (ML) / data mining (DM) workbench [4,13,20] with an ever growing popularity in the research community. Its basic functionalities include but are not limited to: preprocessing (e.g. over 75 data filters and several data sources), classification algorithms (e.g. naive bayes, C4.5, M5, bagging, boosting, etc.), clustering (e.g. k-means, expectation maximization (EM)-based mixture models, etc.) as well as a set of attribute

selection and data visualization methods. Researchers benefit from the availability of a wide range of algorithms to compare against while practitioners can apply these algorithms to their target domain.

WEKA has been designed for "traditional" DM and ML applications, thus it does not natively support a stream-based analysis. It is meant to filter data, extract features and apply learning algorithms on a finite set of data samples. Despite the generally good quality of the code, the class hierarchy and interfaces appear cluttered up. They seem historically grown and their purpose is not always clear.

Massive online analysis (MOA) is an open source software framework that includes a set of online and offline algorithms for clustering as well as classification of evolving data streams. It has been created as an expandable platform where researchers can contribute their algorithms and test them against others [2]. Practitioners can utilize MOA's capabilities to evaluate real world problems on a set of algorithms and choose the best one [2]. MOA is related to WEKA and algorithms in both frameworks can be used interchangeably. It can be utilized to create (and re-use) benchmark settings. Thus enabling researchers and practitioners to create easily repeatable experiments as well as reproducible results.

The general software architecture allows MOA to be easily extended in three ways: (1) data sources / generators, (2) learning algorithms and (3) evaluation methods. In principle, these points also describe the overall workflow within the MOA framework. Thus a rather simple and sequential workflow is implemented in the framework, which represents a common scenario and does make sense in cases where processed data is at hand. However, there is rarely a real world case that does not require at least some preprocessing of data streams. This functionality would come handy in rapid prototyping scenarios. However, this kind of functionality greatly enhances usability and eases prototyping efforts. Additionally, more flexibility regarding the choice of workflows is desirable.

Additional related frameworks and projects exist. These include streams [3], Spark [21] and Storm [11]. While these frameworks are related in the sense that they provide modularized or iterative processing capabilities, they do not meet the envisioned requirements for one reason or another. In particular, some of these are intended for use in clustering environments and utilize a map-reduce type of architecture. This may be effective but limits the computations operations and transformations [21]. Another reason is that most of these frameworks can only deal with sequential information flow (i.e. flow between computational modules) or can not deal with graph-like workflows (i.e. computational modules having both multiple predecessors

as well as multiple successors).

3 Requirements

To overcome the above mentioned drawbacks the following requirements of the MOSIS framework were elaborated.

- Stream-based: Data samples pass through modules and are intended to be inspected once (or not at all). Nonetheless, modules may choose to buffer data samples if time and space constraints allow it.
- Reusability: The reusability of framework components and modules is essential. Each module is intended to perform a single task only.
- Scalability: The framework should support processing of streams that contain large numbers of data samples (possibly infinite). The same applies to stream-based algorithms within the framework.
- Maintenance: A set of design patterns [12] shall be used to ease maintenance and facilitate reusability.
- Flexibility: Information flow among modules can be dynamic (i.e. conditional branching) as opposed to a static (not changing) flow of information.
- Extensibility: Adding new modules and algorithms must be easy.
- Involvement: The community shall be provided with the means to easily add contributions to the framework.
- Portability: The framework and modules shall be usable on a broad range of devices. This includes regular desktop computers, mobile phones, wearables and other devices.

These requirements merely present a high-level outline. Some of them may contradict each other. Thus their benefits and drawbacks need to be evaluated to see which outweighs the other.

4 Design Aspects

MOSIS has been developed with a focus on several design aspects. Here a set of fundamental design decisions and aspects are highlighted.

Likely the most important and equally obvious design aspect relates to the internal representation of a data stream. The stream itself may be arbitrary typed (i.e. a simple sequence of floating point numbers in binary format or a more complex structure such as user datagram protocol (UDP) or transmission control protocol (TCP) packets). This must be carefully considered as every module, every algorithm and every filter, in the MOSIS framework will be influenced by this decision.

The main two options are: (1) create modules that parse streaming data and provide the results to other modules or (2) have every module do the parsing on its own. Latter approach would result in a simple interface for modules. All inputs and outputs could be realized with simple byte-streams (e.g. buffers in main memory, files, queue-like structures, etc.). However, this simplicity comes at the cost of redundant code and fails to adhere reusability and maintenance requirements (i.e. modules perform at least three tasks: parsing stream, processing data and writing formatted results). Additionally, more time on parsing and formatting streams would be spend than absolutely necessary thus slowing things down.

On the other hand, the first option would result in a more complex interface for modules and thus making it more difficult to verify compatibility of succeeding modules (i.e. in terms of input and output data types). However, this approach fits in nicely with the requirements shown in Section 3. In the end, it was decided that the benefits of this approach outweigh its drawbacks. Thus MOSIS employs a mechanism that first splits a continues stream of data into its atomic elements or data samples. These elements are then pushed to the corresponding / succeeding modules. Another benefit is that algorithms do not have to actively wait for data samples, but instead they are activated on demand. Thus they are only used when needed. Furthermore, this approach highlights the stream-based approach as each module is presented with a single data sample at a time, thus online processing is implicitly suggested by the interface.

Another important aspect deals with the representation of information flow. In a simple form, modules could be viewed as nodes in a tree-like structure (i.e. root=data source, leaf=sink). However, this approach fails to support the ability to "merge" two or more streams into a single one. Neither circles (or loops) can be modeled. For those reasons, a graph structure was adopted. Thus each module can have an arbitrary number of inputs (e.g. data samples from various streams) and outputs (e.g. filtered value, processed values, results). Modules, that provide the service of reading a stream and splitting them into usable chunks, are referred to as data sources or data feeds. On the other hand data sinks can be utilized to store or

transport the chunks.

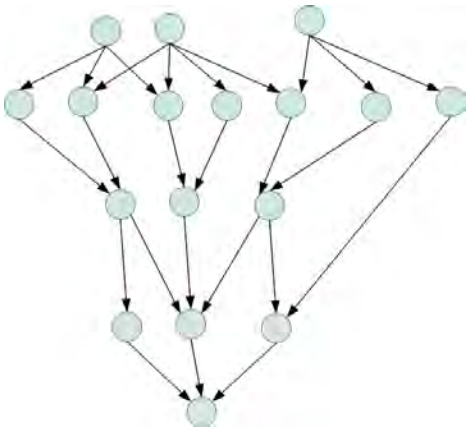


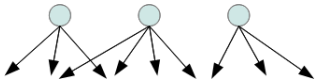
Figure 1: An example demonstrating the flow of data samples within a "sketch" or graph of MOSIS

.

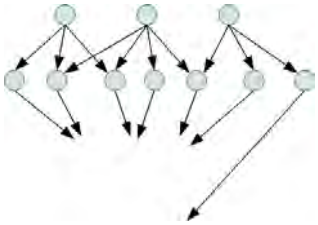
As hinted in Section 3, each module is meant to be self-contained. They do not have knowledge regarding its predecessors and successors. Instead they require a certain type of input data and produce outputs of a certain type. Knowledge about predecessors and successors resides in the design of the information flow or graph / "sketch" (see Figure 1). This eases development and rapid prototyping efforts. Developers can concentrate on the development of their algorithms rather than having to worry about all preprocessing and related matters.

Additionally, the concept of conditional branching is introduced in the MOSIS framework. It can be used to guide data samples through the graph based on whether or not they meet certain conditions. Links between modules can be equipped with the option to choose which data samples they want to transport and which data samples they want to reject (i.e. reject every second data sample, accept only samples above a threshold, etc.).

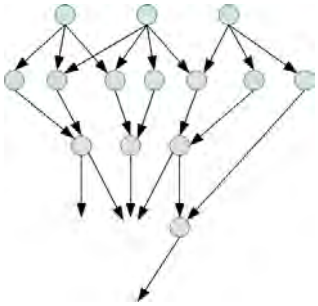
Each module can output an arbitrary number of data samples at a time. Thus enabling modules to reduce and increase sampling frequencies within the graph. Outbound data samples are pushed to all successors. Similarly, each module can have multiple incoming connections. The actual transport among modules follows an iterative deepening approach. The data sources are queried first, then all successor modules are activated. This process continues until a sink is reached (see Figure 2).



(a) Level 0



(b) Level 1



(c) Level 2

Figure 2: Illustrates the progressive deepening while iterating over modules in the graph. A modified level-order iteration is being employed. At first, all data feeds are queried (level 0). In the second step, the next level of modules is queried as well as all data feeds (level 1). Afterwards, modules of the third level are queried as well (level 2).

5 Repositories

MOSIS is meant to thrive on community contributions. For this reason a packaging system is employed that enables easy extension and ensures quality of code. Three repositories are envisioned.

- Core: It includes basic interfaces and modules that make up the framework itself. The core is self-contained and provides a set of tools to create and manipulate "sketches".
- Extra: This is where filters, machine learning and signal processing modules are located. Some dependencies to external libraries may be

introduced.

- Community: Any member of the community can contribute modules (i.e. new algorithms, filters, etc.). These can then be downloaded and used in "sketches".

For maintenance reasons, the "core" and "extra" repositories are solely maintained by a set of dedicated developers and researchers. This decision was made to ensure an adequate level of documentation and quality of code. These restrictions do not apply to the "community" repository. Here all members can create modules and share them with the community. This does not exclude these modules from ever being part of the "core" or "extra" repositories. Instead it is intended to speed-up the process of integration. These modules are already available even before they have been integrated (i.e. formatted and documented) in the official framework.

6 Status Quo

Currently, MOSIS provides a simple application programming interface (API). It can be utilized to construct and execute graphs within the framework. The framework supports several streaming sources / sinks (e.g. files, network connections, etc.) as well as a set of formats (e.g. plain text, CSV, binary, etc.). Additionally, a set of artificial data generators are available (e.g. mathematical functions, noise). Apart from the input / output handling, a small set of algorithms is already included in MOSIS.

Furthermore, MOSIS is capable of pushing data samples through graphs / "sketches" in an orderly and repeatable fashion. Modules are activated as they are needed and can have any input / output data type. It can handle circles within graphs and can dynamically increase / decrease sampling frequencies.

At the time of writing a set of data generators (i.e. STAGGER Concepts Generator [17], LED Generator [5]), classification algorithms (i.e. Hoeffding Tree [8], naive bayes, bagging, boosting), clustering approaches (i.e. CluStream [1], D-Stream [19]) and stream mining methods (i.e. PC [10], PCSA [10], LogLog [9], Count-Min [7]) as well as several filters and feature selection methods are being realized in MOSIS.

The source code of MOSIS is publicly available on GitHub at [14].

7 Future Work And Conclusions

This paper introduced the design concept of a framework for signal processing, analysis and machine learning. The requirements, several design aspects and its status quo have been discussed.

MOSIS bridges the gap between the fields of machine learning and signal processing. Upon completion, it will provide a set of algorithms from both worlds. Furthermore, it includes benchmarking abilities to ease evaluation of stream learning algorithms and provide reproducibility. MOSIS is open source and is extensible in many ways. The framework hopes to thrive on community contributions.

The framework is intended to be used by researchers and practitioners alike. Researchers can use MOSIS to evaluate newly developed algorithms against a set of competitive solutions without having to implement them themselves. Enthusiasts can utilize MOSIS as a learning platform for signal processing and stream algorithms. Practitioners can use it to find a suitable algorithm to their problem. It is not intended to replace existing implementation, but rather provide broader access to these methods in order to kick-start development, learning and evaluation. Its simple structure allows utilization in classroom scenarios.

The current version of MOSIS utilizes a simple API. However, it is planed to include a command line interface as well as a graphical user interface. Latter interface should improve usability and further reduce rapid prototyping efforts of streaming algorithms. At the time of writing, the collection of included algorithms is being expanded for a more complete set of algorithms. A set of show cases is being prepared as well.

Acknowledgments

This work has been motivated and partly funded by the European Commission through AAL JP Call 1 project HELP and ICT FP 7 project REMPARK (287677). The authors acknowledge the support by the Commission and the HELP and REMPARK partners for their fruitful work and contribution in the research.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for Clustering Evolving Data Streams. In *VLDB*, pages 81–92, 2003.

[2] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering. *Journal of Machine Learning Research - Proceedings Track*, 11:44–50, 2010.

[3] C. Bockermann and H. Blom. The Streams Framework. Technical Report 5, TU Dortmund University, 12 2012.

[4] R. R. Bouckaert, E. Frank, M. A. Hall, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. WEKA–Experiences with a Java Open-Source Project. *Journal of Machine Learning Research*, 11:2533–2541, 2010.

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

[6] B. T. Cole, S. H. Roy, C. J. De Luca, and S. H. Nawab. Dynamic Neural Network Detection of Tremor and Dyskinesia from Wearable Sensor Data. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6062–6065, 31 2010-September 4 2010.

[7] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *Journal of Algorithms*, 55(1):58–75, April 2005.

[8] P. Domingos and G. Hulten. Mining High-speed Data Streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

[9] M. Durand and P. Flajolet. Loglog Counting of Large Cardinalities. In G. Battista and U. Zwick, editors, *Algorithms - ESA 2003*, volume 2832 of *Lecture Notes in Computer Science*, pages 605–617. Springer Berlin Heidelberg, 2003.

[10] P. Flajolet and G. N. Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[11] E. Frachtenberg, F. Petrini, J. Fernandez, and S. Pakin. STORM: Scalable Resource Management for Large-Scale Parallel Computers. *IEEE Trans. Comput.*, 55(12):1572–1587, December 2006.

[12] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[14] MOSIS. Modularized Signal Processing and Analysis, January 2015. http://www.github.com/claasahl/MOSIS.

[15] S. H. Roy, B. T. Cole, L. D. Gilmore, C. J. De Luca, and S. H. Nawab. Resolving Signal Complexities for Ambulatory Monitoring of Motor Function in Parkinson's Disease. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4832–4835, 30 2011-September 3 2011.

[16] A. Salarian, H. Russmann, C. Wider, P. R. Burkhard, F. J. G. Vingerhoets, and K. Aminian. Quantification of Tremor and Bradykinesia in Parkinson's Disease Using a Novel Ambulatory Monitoring System. *IEEE Transactions on Biomedical Engineering*, 54(2):313–322, February 2007.

[17] J. Schlimmer and J. Granger, RichardH. Incremental Learning from Noisy Data. *Machine Learning*, 1:317–354, 1986.

[18] M. G. Tsipouras, A. T. Tzallas, G. Rigas, P. Bougia, D. I. Fotiadis, and S. Konitsiotis. Automated Levodopa-induced Dyskinesia Assessment. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2411–2414, 31 2010-September 4 2010.

[19] L. Tu and Y. Chen. Stream Data Clustering Based on Grid Density and Attraction. *ACM Trans. Knowl. Discov. Data*, 3(3):12:1–12:27, July 2009.

[20] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning, Tools and Techniques*. Morgan Kaufmann series in data management systems. Elsevier/Morgan Kaufmann, Amsterdam [u.a.], 3. ed. edition, 2011. XXXIII, 629 S. : Ill., graph. Darst.

[21] 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 *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.