Parallel Concept Drift Detection with Online Map-Reduce

Artur Andrzejak
Heidelberg University
Heidelberg, Germany
artur.andrzejak@informatik.uni-heidelberg.de

João Bártolo Gomes
Institute for Infocomm Research (I2R)
*A*STAR, Singapore*
bartologjp@i2r.a-star.edu.sg

***Abstract*—Empirical evidence shows that massive data sets have rarely (if ever) a stationary underlying distribution. To obtain meaningful classification models, partitioning data into different concepts is required as an inherent part of learning. However, existing state-of-the-art approaches to concept drift detection work only sequentially (i.e. in a non-parallel fashion) which is a serious scalability limitation. To address this issue, we extend one of the sequential approaches to work in parallel and propose an Online Map-Reduce Drift Detection Method (OMR-DDM). It uses the combined online error rate of the parallel classification algorithms to identify changes in the underlying concept. For reasons of algorithmic efficiency it is built on a modified version of the popular Map-Reduce paradigm which permits for using preliminary results within mappers. An experimental evaluation shows that the proposed method can accurately detect concept drift while exploiting parallel processing. This paves the way to obtaining classification models which consider concept drift on massive data.**

***Keywords*-Concept-Drift; Parallel Data Mining; Map-Reduce**

I. Introduction

Very large data sets are rarely (if ever) stationary. For example, Peter J. Huber reports in [16, p.12] that the largest homogeneous data set with a stationary distribution he encountered in several decades of his carrier contained barely 9000 samples. [18] reports that models used in a direct mail campaign needed to be re-trained at least every few *weeks* for good accuracy, while the initial (and devastating for accuracy) assumption on model's validity were months. Such effects can be explained by the fact that the sampled processes evolve over time or the underlying parameters are changed.

The implications for concept learning on massive data sets are manifold. The good news is that even if there exist a stationary *and* very large data set, taking a random subsample of it will yield a data set with identical distribution but manageable size [16, p.52]. The backside is that learning on very large data sets should inherently incorporate concept drift detection. Consequently, this sheds some doubts about utility of many recent machine learning methods which are plain "translations" of traditional (small-data) approaches to parallel or distributed setting [5], [3]. Unfortunately, hardly any of the contributions in this currently very active field consider the issues of concept drift.

For practical purposes, learning - and especially classification - in large-data scenario should produce *multiple* models (each for a different identified concept) along with information to which part of the data each model belongs to. This approach has been already studied in context of data streams and/or data sets of traditional size and several excellent solutions have been proposed [17], [9]. For handling very large data sets, parallel and/or distributed processing has become a de-facto standard, and so an extension of the above methods is desirable. This work investigates how to transfer and adapt the above sequential approaches to classification with concept drift detection to the setting of parallel/distributed processing.

We focus on a particular distributed programming paradigm, namely Map-Reduce. Popularized by Google, this programming model has become an economical choice for fault-tolerant and massively parallel data processing [7]. Multiple sequential machine learning approaches have been adapted to Map-Reduce [5], resulting in highly scalable algorithms.

The drawback of established Map-Reduce implementations such as Hadoop is that they work only in batch mode. They do not allow stream processing nor exploiting of preliminary results (especially not in the Map-Phases). As it will become obvious in Section III-A, the latter property prevents an efficient implementation of state-of-the-art concept drift detection methods such as in [8].

To mitigate this problem, we use a modified version of Map-Reduce which works in an online fashion and permits to use preliminary results. Such systems been proposed independently in [6] and [4]. In addition to publishing preliminary results, framework in [4] also allows to "send back" such results to mappers. This facilitates efficient implementation of iterative algorithms such as k-means clustering, as shown in [4]. This property is also essential for efficient switching to new classification models once a concept drift is likely.

The major contribution of this work is an Online Map-Reduce Drift Detection Method (OMR-DDM). It uses the combined online error-rates of several classification models (running on different mappers of a Map-Reduce framework) to identify changes in the underlying concept. While the mappers perform classification on parts of a data stream and verify the correctness on each instance of the stream, the reducer combines this output and determines the current model state. The appropriate signals (normal, warning, drift) are sent back to mappers which create new models (and save old ones) if appropriate. Our method is complemented by a mapper-reducer synchronization strategy and evaluated in terms of accuracy and runtime performance.

The rest of the paper is organized as follows. In Section II we formally define the problem of concept drift detection and describe a sequential solution. Section III describes the proposed Online Map-Reduce Drift Detection Method (OMR-DDM) which is evaluated empirically in Section IV. The related work is discussed in Section V. We conclude with Section VI where we also outline our plans for future work.

II. Problem Definition

Let X be the feature space for the explanatory variables and Y be the space of the target variable (i.e., the space of class labels). We denote by $f : X \rightarrow Y$ the underlying function to be learned. Learning is performed online on a *data stream*, i.e. an ordered list of records $(x \in X, y \in Y)$.

We assume that the data stream can be partitioned into sequences $S_1, S_2, ..., S_n$ where each S_i is a sequence of records generated by some stationary distribution D_i. Each D_i can be described by a single underlying function f_i (i.e. S_i corresponds to a stable concept). If $f_i \neq f_{i+1}$, there is a *concept drift* between S_i and S_{i+1}. It results from a change in the posterior distribution of the class membership $P(Y|X)$. Such a drift might be accompanied by changes in the distribution of class prior $P(Y)$, the unconditional feature distribution $P(X)$, and the feature distribution conditioned on classes $P(X|Y)$.

The major issue is to find the correct partitioning of the data stream such that each a distribution on a sequence S_i can be accurately described by a single underlying function f_i and $f_i \neq f_{i+1}$. As in the online (one-pass) learning scenario we cannot "go back" to previous parts of the data stream, we need to detect distribution changes in one pass over data. The following approach is used here (see Sec. II-A). If for each concept the sequence S_i contains a large number of records, it is be possible to adapt the current model f_i to D_i. Concurrently with this incremental learning we check online the accuracy of f_i on the incoming data stream. A *decreasing* accuracy is a likely indication of a concept drift; the algorithm signals this and switches to a model learned on more recent data. The challenge studied in this work is to adapt this schema to the setting of (massively) parallel computing framework, namely Map-Reduce.

An additional issue is that in real problems there could be a transition phase between two consecutive sequences S_i and S_{i+1} where some records of both distributions occur. A record generated by a distribution D_{i+1} is noise for D_i and vice-versa. This is an additional issue for change detection algorithms to address, as they must differentiate noise from change. The difference between noise and records of another distribution is persistence: there should be a consistent set of records of the new distribution. Therefore, change detection algorithms must combine robustness to noise with sensitivity to concept changes.

A. Drift Detection

The goal of our approach is to identify occurrence of a concept drift while a dataset or a stream is processed in parallel. For this purpose we use the error-rate of the learning process, similarly to what has been proposed for the sequential case by Gama et al. in [8]. The learning rate is used to obtain the *(model) confidence level* which determines a concept drift adaptation action. The confidence levels tell us to do nothing ("normal"), to prepare a new model ("warning") or to switch to the previously prepared model ("drift").

Our setting is that the data stream is used both for concept learning and testing of classifier accuracy. In detail, each incoming record $r = (x, y)$ is first used to test the accuracy of the current model f by computing the boolean $c_r := f(x) = y$ (i.e. c_r is true if $f(x) = y$). Then r is incrementally incorporated into the model. The subsequent values of c_r constitute a *correctness vector* (CV). The learning process can be in a state where stable concepts (i.e. with stationary distribution) are observed followed by changes leading to a new period of stability with a different underlying concept.

The binomial distribution gives the general form of the probability of observing an error and so the error-rate of the learning algorithm is a random variable from a sequence of Bernoulli trials. For each record i in the data stream and $error_i$, i.e. the number of mis-classifications until i, the error-rate is $p_i = (error_i/i)$. This can be interpreted as the probability of misclassifying an instance; its standard deviation is given by $s_i = \sqrt{p_i(1 - p_i)/i}$.

It is assumed that p_i will decrease while i increases if the distribution of the examples is stationary. A significant increase in p_i indicates that the class distribution is changing. The values of p_i and s_i are calculated incrementally and their minimum values (p_{min}, s_{min}) are recorded when $p_i + s_i$ reaches its minimum value. Using the above metrics, we can define two abnormal confidence levels and the corresponding adaptation strategies:

- *Warning level* occurs if $p_i + s_i \geq p_{min} + 2\,s_{min}$ (95% confidence). At or beyond this level the incoming records are stored in anticipation of a possible change in concept.
- *Drift level* occurs if $p_i + s_i \geq p_{min} + 3\,s_{min}$ (99% confidence). Beyond this level the concept drift is considered to be detected and the adaptation strategy consists in resetting the model induced by the learning method along with the values for p_{min} and s_{min}. Furthermore,

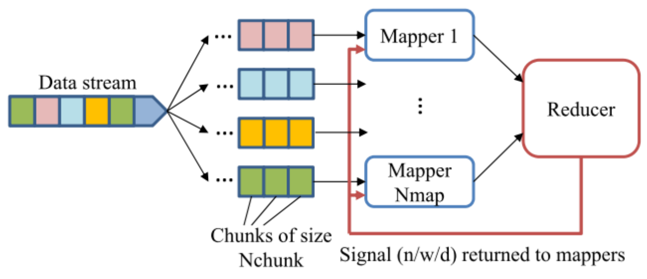

Figure 1. Architecture of the Online Map-Reduce Drift Detection Method

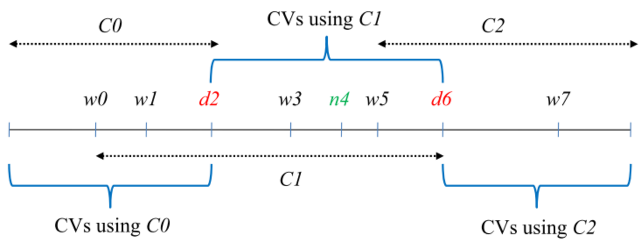

Figure 2. Mapper view of the learning process

the strategy uses the records stored since the oldest warning level (but after last drift or normal level) to learn a new model of the current target concept.

III. Map-Reduce Drift Detection Method

A. System Architecture and Deployment Scenarios

Our approach uses the Map-Reduce programming paradigm [7] which has become the tool of choice for fault-tolerant distributed processing of massive data sets. In the first step the input data is partitioned and processed independently by map tasks, with each one emitting a list of <key, val> pairs as output. Subsequently these pairs are grouped by keys (by the framework) which gives (for each unique key k) a list of corresponding val's emitted by all mappers. In the last step the "per-key" lists of val's are processed independently by reduce tasks, which collectively creates the final output. Note that many algorithms require multiple such phases to complete.

The general idea of using Map-Reduce for the concept drift method outlined in Sec. II-A is illustrated in Figure 1. The general learning process can be divided into the following steps:

- The data stream is split ("demultiplexed") into *chucks* of size Nchunk and passed to each of the Nmap mappers in a round-robin fashion.
- Each mapper i uses a data stream classification algorithm to learn on "its" incoming records and outputs regularly the locally computed correctness vector CV_i.
- The reducer combines periodically the correctness vectors $CV_1, \ldots, CV_{\text{Nmap}}$ and outputs *signals* that represents the model confidence level (i.e., normal, warning, drift) additionally labeled by the stream position.
- The signals are passed back to the mappers which act on it according to the adaptation strategy described in Sec. II-A.

To implement the last of the above steps we need to pass data from the reducer back to the mappers. Unfortunately, traditional Map-Reduce implementations work only in batch mode and do not allow using preliminary results in the mappers. In such implementations we would need to use a sequence of Map-Reduce phases where each reducer terminates the phase when a drift signal is detected and passes the previously detected warning signals to the mappers of the next phase. Obviously, this approach would be very inefficient. As a remedy we utilize an online version of Map-Reduce [1] which allows to access the preliminary processing results in the mappers (see [4] for architectural details).

Note that we do not necessarily need a "demultiplexer" which splits the data stream into chunks and sends them to mappers. It is sufficient to replicate the data stream to each mapper M use there a small filter which retrieves only chunks relevant to M. While this implementation increases the overall data transfer cost, it eliminates the need for a single node which has to process the whole stream and might become a bottleneck.

B. Mappers

As illustrated in Figure 1 the mappers process the records assigned to them and learn their classifiers incrementally and independently. They also check regularly whether the reducer has generated any new signal (normal, warning or drift) or not. When a signal is generated the following actions are performed:

- The first *warning* signal after start, a drift-signal, or a normal/stationary signal triggers a creation of a new instance of the base learner while a snapshot of the current model is saved provisorily (so that the snapshot represents only the concept from S_i). Since then the new model is trained concurrently with the current one. In case of a false alarm (i.e., warning followed by normal signal) the new model and the provisorily saved snapshot of current classifier are discarded. In a series of warnings the oldest warning is considered (i.e., the system continues in the warning state).
- When *drift* is signaled the classifier that was learning the new concept after a warning is now used to represent the current concept. The snapshot of the last classifier is now persistently saved (using the copy that was created at the time of warning, so no records from S_{i+1} are included). The values of p_i, s_i, p_{min}, s_{min} are all reset.

Figure 2 gives an example of the learning process in a mapper. It shows the signals normal/stable, warning and drift

($n4$, $w0$, $w1$, $d2$, $d6$) with their positions in the data stream, the created classifiers ($C0$, $C1$, $C2$) and which classifiers have been used to generate the correctness vectors (CVs).

C. Reducer

The reducer collects the correctness vectors CVs sent regularly by each mapper and buffers them separately for each mapper, see Figure 3. Once the CVs with length at least Nchunk each has been received from each mapper, the reducer multiplexes them by assembling a "merged" CV which corresponds to consecutive positions in the data stream (note that its length is at least Nmap $\cdot$ Nchunk). The reducer evaluates then the merged CV, updates the results (i.e., p_i, s_i, p_{min}, s_{min}), generates the signals and sends these back to the mappers.

In an early implementation we have experienced severe synchronization problems: some mappers were processing too much data while reducer or other mappers did not receive enough CPU cycles. To solve this issue we implemented and evaluated two approaches. The first one uses a *decision frontier* (DF) defined as the latest stream position for which the CVs from all mappers have been received and processed as described above. The mappers receive regularly the DF and do not process the input stream beyond DF plus a certain threshold (usually Nmap $\cdot$ Nchunk).

It may also happen that some mappers are particularly slow. In that case, if the difference between latest stream positions of fastest mapper p_{max} and position of slowest mapper p_{min} is greater than a maximum distance D, then the errors are updated ignoring the slowest mapper (in the figure CV_0 is ignored).

The second synchronization strategy is to let mappers wait for a signal *go-on* from the reducer after a chunk (i.e. Nchunk samples) has been processed and its CV is send to the reducer. The reducer sends *go-on* when it has obtained CVs from all mappers. In other words, after each (collective) processing of chunks by all mappers the processing of these chunks by the reducer must take place (while mappers are suspended until reducer finishes). While this is computationally less efficient approach, it has turned out to be much more stable in terms of accuracy and is used in the evaluation section.

IV. Experimental Results

We conducted experiments to evaluate accuracy and run-time performance of the method proposed in this paper. The algorithms were coded in Java and used as a run-time a shared-memory Map-Reduce implementation which is a part of the MRStreamer framework [1]. MRStreamer provides a basic Apache Hadoop-compatible API and some additional features, notably possibility to use preliminary results in the mappers (online processing).

We also used (modified) classes of the MOA [15] environment, especially `SingleClassifierDrift` and `DDM`.

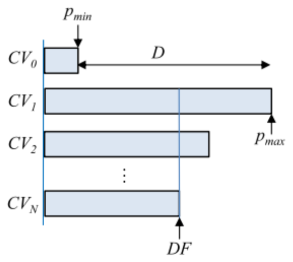

Figure 3. Merging of CVs in the reducer and a synchronization strategy using the decision frontier (DF)

MOA [15] stands for Massive Online Analysis and is an open-source framework with tools for data stream learning problems.

The results have obtained on JVM 1.6.0_26 running under Debian 6.0.5 (64 bit, 2.6.32-5-amd64) on a KVM virtual machine with 8 GB RAM (2.5 GB JVM heap space) and I7-2600 host processor (4 cores and 4 hyper-threading units).

A. Datasets

The *SEA* dataset was introduced by Street and Kim [19] and is considered a concept drift benchmark. The dataset has two classes $c0$, $c1$ and three features with values between 0 and 10 (only the first two f_1, f_2 are relevant). The target concept function classifies a record as $c1$ if $f_1 + f_2 \leq \theta$, otherwise as $c0$. MOA package allows to generate this dataset while controlling the threshold parameter θ for each concept as well the amount $p\%$ of noise (defined as percentage of records where the class label is flipped). We have used 10% noise and θ values of 8, 9, 7 and 9.5 as proposed in [15]. The concepts have been changed at positions 15,000, 30,000 and 45,000, yielding 60,000 records in total.

The *Usenet* dataset is a simulation of news filtering with a concept drift [10]. The different concepts are related to the interest of a user that changes over time. For each concept the simulated user is subscribed to mailing list of four topics and is interested in two of these. Attribute are represented as bag of words with a total of 659 attributes and 5,931 records. The concepts change at positions 1,192, 2,377, 3,562, 4,354 and 5,143.

B. Accuracy Evaluation

In our study we varied two parameters: the number of mappers Nmap and the size of a data stream chunk Nchunk delivered to a mapper (which is also chunk size processed by it before waiting for reducer's reaction) with Nmap $= \{1, 2, 4, 6, \ldots, 20\}$ and Nchunk $= \{1, 2, 4, 8, 12, 16, 20, 32, 40, 64, 128, 256, 384, 512, 768, 1024\}$. For each value combination we execute the experiment 5 times and report averaged values.

Each parameter introduces performance vs. accuracy trade-offs: (i) larger Nmap values mean larger degree of parallelism, but it also means that each mappers receives

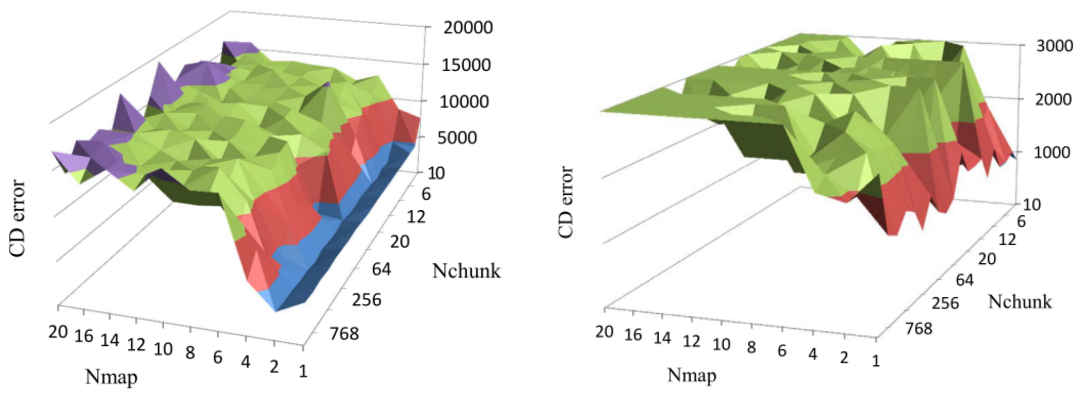

Figure 4. Error of concept drift detection expressed as sum of absolute distances between the true and the detected concept drift positions (with penalties for not detected positions). Left: SEA dataset; Right: Usenet dataset

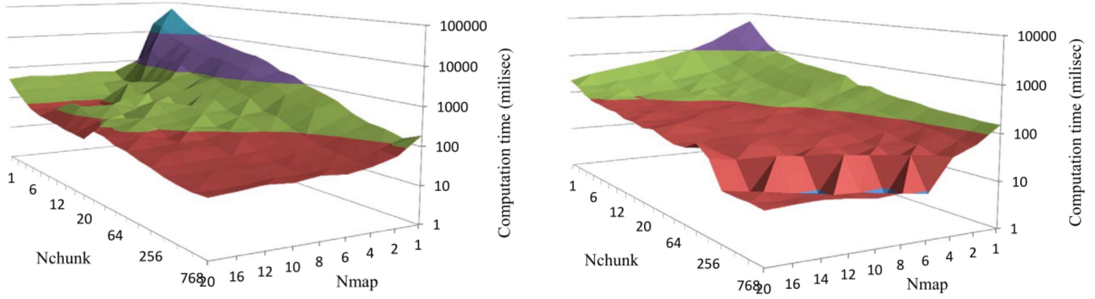

Figure 5. Execution time of the algorithm in milliseconds. Left: SEA dataset; Right: Usenet dataset

less data of a single concept (namely a fraction $1/(\text{Nmap})$ of the sequential version); (ii) larger Nchunk values decrease the number of mapper-reducer synchronization barriers (and so lower mapper's waiting time), but it also increases the delay until a mapper receives a warning/drift signal from the reducer. This delay is proportional to the size of a possibly outdated correctness vector delivered to the reducer.

As the error metric for the concept drift detection we used the sum of absolute distances between the true and the detected concept drift (CD) positions. We have to treat two special cases: (i) if there are several (different) detected positions close to one true CD position, we use only the detected position with *largest* distance from the true one; (ii) if a true CD position has not been detected at all, we add a fixed penalty to the sum (for each such case). We set this penalty to roughly $1/3$ of the distance between two true CD positions, i.e. 5,000 and 400 for SEA/Usenet datasets.

Figure 4 shows the detection errors for both datasets. Obviously the larger number of mappers Nmap has a detrimental effect on accuracy, while the chunk size Nchunk has hardly any effect. We explain it by the mentioned fact that with larger Nmap map's classifier receives less data of a single concept. Interestingly, using two mappers yields better results for the SEA dataset, and 4-6 seem to be worse than more mappers for the Usenet dataset.

C. Scalability Evaluation

Figure 5 shows the execution times for both datasets in milliseconds (logarithmic z-axis). A larger number of mappers consistently reduces this time, with the effect leveling out at Nmap = 8 (number of virtual CPUs of our machine). Increasing the chunk size Nchunk reduces the execution time even more strongly, which can be attributed to lower number of synchronization barriers. To yield an optimal balance between accuracy and execution time, a large chunk size (above 128) combined with low number of mappers (2 or 4) should be used in case of these datasets.

V. Related Work

Processing of massive data. Most such distributed processing in business domains (and outside High-Performance Computing) is performed on clusters of desktop-class machines. Here the Map-Reduce programming model [7] became the most used approach. Follow-up open-source frameworks became very popular, especially Apache Hadoop. The extensions to online processing are Hadoop Online system [6] as well as MRStreamer [1], [4].

Parallel and distributed data mining. Interest in parallel and distributed machine learning exists since more than two decades, with a variety of contributions in unsupervised and supervised learning. While these solutions can be considered as independent and sometimes platform-specific, some recent works attempt to unify the approaches either via theory such as sufficient statistics or via common programming paradigms such as Map-Reduce. For example, [5] outlines how 10 popular algorithms can be ported to Map-Reduce and the project Apache Mahout provides real implementations under this programming paradigm.

Signal drift detection. In the context of sequential learning Gama et al. [8] approach monitor the error-rate of the learning algorithm to find drift events. Alternatively Baena et al. [2], uses the distribution of the distances between classification errors to signal drift. If the distance (resulting from more consecutive errors) is above a pre-defined threshold, the underlying concept must be changing and a change event is triggered. The basic adaptation strategy after drift is detected is to discard the old model and learn a new one to represent the new underlying concept [8], [2], as both methods are proposed in the context of online learning.

VI. Conclusion

We propose a Online Map-Reduce Drift Detection Method (OMR-DDM) for scalable classification on massive data sets where underlying distribution is non-stationary. Our method uses the combined online error-rate of the parallel classification algorithms to identify concept changes.

We have performed experiments to test the effectiveness and efficiency of OMR-DDM on two datasets. The results show that the running time decreases with the degree of parallelism, however there is a trade-off in terms of accuracy. However, with low number of mappers and chunk sizes (number of samples processed by a mapper in a single step) above 100 our method achieves accuracy similar to the sequential version and has low running time.

In our future work we plan to evaluate other drift detection methods [2], recurring concept methods [13], [12], [14] and a collaborative method [11] in the streaming Map-Reduce framework. We also we plan to explore strategies to visualize the results and combine the obtained classifiers.

Acknowledgment

We like to thank João Gama and Shonali Krishnaswamy for their helpful comments and discussion, and Udit Halder for a preliminary version of the code.

References

[1] PVS Group at Heidelberg Univ. MRStreamer, 2011. http://pvs.ifi.uni-heidelberg.de/software/mrstreamer/.

[2] Manuel Baena-García, José del Campo-Ávila, Raul Fidalgo, Albert Bifet, Ricard Gavaldà, and Rafael Morales-Bueno. Early drift detection method. In *4th Int. Workshop on Knowledge Discovery from Data Streams*, 2006.

[3] Ron Bekkerman, Mikhail Bilenko, and John Langford, editors. *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2012.

[4] Joos-Hendrik Bose, Artur Andrzejak, and Mikael Hogqvist. Beyond online aggregation: Parallel and incremental data mining with online mapreduce. In *ACM Workshop on Massive Data Analytics over the Cloud (MDAC 2010)*, Apr 26 2010.

[5] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288, 2006.

[6] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. MapReduce Online. Technical Report UCB/EECS-2009-136, Oct 2009.

[7] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *CACM*, 51(1):107–113, 2008.

[8] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. *Advances in Artificial Intelligence–SBIA 2004*, pages 66–112, 2004.

[9] J. Gama, R. Sebastiao, and P. Rodrigues. Issues in evaluation of stream learning algorithms. In *15th ACM SIGKDD*, pages 329–338. ACM New York, NY, USA, 2009.

[10] João Gama. Datasets for concept drift, 2010. http://www.liaad.up.pt// kdus/kdus_5.html.

[11] J.B. Gomes, M. Gaber, P. Sousa, and E. Menasalvas. Context-aware collaborative data stream mining in ubiquitous devices. *Advances in Intelligent Data Analysis X*, pages 22–33, 2011.

[12] J.B. Gomes, E. Menasalvas, and P.A.C. Sousa. CALDS: context-aware learning from data streams. In *Proc. StreamKDD'10*, pages 16–24. ACM, 2010.

[13] J.B. Gomes, E. Menasalvas, and P.A.C. Sousa. Tracking recurrent concepts using context. *Rough Sets and Current Trends in Computing*, pages 168–177, 2010.

[14] J.B. Gomes, E. Menasalvas, and P.A.C. Sousa. Learning recurring concepts from data streams with a context-aware ensemble. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 994–999. ACM, 2011.

[15] G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis, 2007 - http://sourceforge.net/projects/moa-datastream/.

[16] Peter J. Huber. *Data Analysis - What Can Be Learned From the Past 50 Years*. John Wiley & Sons, 2011.

[17] I. Koychev. Gradual forgetting for adaptation to concept drift. In *Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning, Berlin*, pages 101–107, 2000.

[18] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999.

[19] W.N. Street and Y.S. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *7th ACM SIGKDD*, pages 377–382. ACM New York, NY, USA, 2001.