Towards Real-Time Machine Learning

Andreas Hapfelmeier[1], Christian Mertes[1], Jana Schmidt[1], and Stefan Kramer[2]

[1] Department of Computer Science, Technische Universität München, 85748 Garching, Germany,
{andreas.hapfelmeier, christian.mertes, jana.schmidt}@in.tum.de,
[2] Institute of Computer Science, Johannes Gutenberg-Universität Mainz, 55128 Mainz, Germany,
kramer@informatik.uni-mainz.de

Abstract. Mining data streams has become an important topic within the last decade, and numerous approaches have addressed various issues in this domain. However, little work has been devoted to the setting where high-speed data streams are potentially faster than the underlying learning algorithm, and nevertheless a prediction needs to be given promptly for each unlabeled instance in the stream. Additionally, a model has still to be trained on the labeled instances of the stream in parallel. To solve this problem, we propose a framework and controller-based approach that ensures proper training of a model and a prediction for each unlabeled instance in the stream. The approach can also be used for data streams with altering stream speeds. We show that the method performs well under heavy system load on both synthetic and real-world data using different learning algorithms. We believe that this type of real-time machine learning, the synthesis of machine learning and real-time systems, raises a number of interesting questions for machine learning research.

Keywords: Data stream mining, assured prediction, real-time systems

1 Introduction

Over the past few years, the amount of collected information has been increasing extremely, and new challenges are posed to classical machine learning algorithms. Web applications, social services and sensors capturing the environment with increasing quality produce a steadily growing mass of data. Next generation sequencing (NGS) technologies double their sequencing capacity of base pairs (bp) per dollar every fifth month [1], the "LifeShirt" project [2] monitors the health status of patients with body sensors, generating 200 MB over 24 h for one person, and the NASA Earth Observation System (EOS) produces 2.9 TB data per day [3], to name just a few prominent examples. Such high speed data streams (DS) can be found in many other areas beyond science as well, like finance, web applications or telecommunications.

While the mass of data is steadily increasing and data streams constantly gain in speed, online learning algorithms used to process the data are naturally limited by their maximal instance processing speed. As the evolution of these massive data streams is much faster than the improvement of CPU power after Moore's law [4], the gap increases between available and processable data. As a consequence, not all provided instances in a stream can be used by the learning algorithm and some have to be skipped. This could be especially harmful if the skipped instances would reveal important insights to the user. To avoid skipping instances and to still enable (potential) insights, currently not processable instances could, in principle, be externally stored for later processing. However, as the data stream speed is higher than the processing speed, the algorithm is constantly challenged by the amount of data, and the amount of stored instances is constantly increasing. Besides memory usage, the time span from the arrival of the instances to their processing (response time) is steadily increasing as well. This processing delay can result in outdated information, and important events might be missed. To address these issues, this paper introduces PAFAS (Prediction Assured Framework for Arbitrarily Fast Data Streams), a framework to handle high-speed data streams that potentially go to or beyond the limits of the processing speed of the online learning algorithm. The contributions of PAFAS are:

1. All unlabeled instances in the data stream receive a prediction.
2. The prediction is given promptly after the instances' arrival time.
3. The prediction model is constantly improved (independent of the DS speed).
4. No external instance storage is needed.

The framework can be applied whenever events have to be detected as soon as possible, and no information is allowed to be missed to detect these events. We believe that concepts for the embedding of machine learning into real-world systems are required, taking into account the time it takes to make a prediction as well as the time it takes to train or refine a model. In this paper, we discuss one such framework and present evidence from experiments with varying loads.

This paper is organized as follows. First, related work is presented. Then, the problem setting is presented along with the proposed framework. Subsequently, the evaluation of the framework is presented in Section 4. The paper closes with a discussion.

2 Related Work

Data stream mining has developed considerably in the past decade and attracted many researchers to adopt existing algorithms for the challenging task to process and reason about instances received at a very high speed [5]. One part addresses the adaptation of batch algorithms to cope with the data stream setting [6] by, e.g. incremental batch approaches [7]. To provide a specific environment for efficient data stream processing, data stream management systems (DSMS) have been developed. Such systems are adaptations of database management systems

(DBMS) to query continuous, unbounded data streams possibly in combination with pre-stored, fixed datasets. Two well-known DSMS, AURORA [8] and STREAM [9], use their own language to query data streams. Both systems also address the problem of too fast data streams, i.e., when the system is not capable of processing all of the instances provided by the data stream. They use *load shedding* (also implemented in a system environment [10]) to select instances of the data stream that should be processed. Based on Quality-Of-Service (QoS) specifications, the system decides which instances are useful for the system to fetch and which instances can be discarded. The main idea is to select instances that will most probably lead to a good prediction. Another possibility to cope with too fast data streams is *sampling*. Sampling is a technique to represent a larger dataset by a smaller selected subset. It was frequently applied to reduce the overall processing time of data mining algorithms and to efficiently scan large datasets [11]. In the simplest case it selects a random subset from the whole data set as an input for the learner. Frequently, the purpose of this is to estimate the quality of the result [12]. Another possibility to cope with very fast data streams is to adapt the mining technique corresponding to the currently available resources. Such methods are summarized under the heading of *granularity-based techniques*. While load shedding and sampling change the input granularity of the data mining method, the output of the data mining method can also be reduced, e.g. the number of rules or clusters [13]. Then, the model that is used for classification is smaller and thus also more time-efficient, i.e., more instances can be processed in less time. This method termed Algorithm Output Granularity (AOG) can also be applied on various data mining schemes like clustering, classification or frequent set mining. Last, anytime algorithms are also often used for altering data stream speeds, as they can be interrupted anytime to return an intermediate result [14]. The more time available, the better the result has to be. Most of the presented approaches make use of a resource monitor (also called controller) that decides how an instance will be processed, depending on the current data stream speed. We will also make use of this successful concept in our work. However, none of these methods addresses the case when there are labeled and unlabeled instances in the data stream and the user expects a classification for each unlabeled instance. If one applies load shedding or sampling on such a data stream, instances may drop out of the process and no prediction would be made for them. If AOG was used in such a case, then still the data stream may be too fast for even the smallest model. This would either lead to a memory exception or long response times for such instances. Anytime algorithms need an initialization period for each instance and consequently, they can be overwhelmed by fast data streams as well. Therefore, we propose an approach that guarantees prompt prediction of each unlabeled instance by adaptation to data streams of varying speeds.

3 Prediction Assured Framework for Arbitrarily Fast Data Streams (PAFAS)

This section first introduces the problem setting and then specifies PAFAS, which is proposed to tackle the problem.

3.1 Problem Setting

A data stream $DS = \{i_1, \ldots, i_j, \ldots, i_\infty\}$ is a possibly unbounded sequence of instances $i \in \mathbb{R}^k$ observed in increasing order of index j, where each instance is observed at a specific time point t_j. Each instance $i_j =< x_{j1}, \ldots, x_{jk-1}, y_j >$ consists of attributes with known values (x_{jk}) and an attribute of interest (y_i, the target variable) with a possibly missing value. Depending on the attribute of interest, we distinguish between two types of data streams. In the first data stream type, the value for the attribute of interest is given (DS_L / labeled data stream) and in the second, the attribute value is missing (DS_U / unlabeled data stream). As the value of y_j in DS_U is important in the application domain, a model M is trained on DS_L, where y_j is known for each instance. Model M is then applied on DS_U to make a prediction $\hat{y_j}$ for y_j. For simplicity, model and learning algorithm are merged into one entity in our framework, incorporating both the representation of the model (function) and learning / adaptation functionality. Each data stream has a specific speed $\boldsymbol{v}_{DS}$, defined as the number of instances observed in the streams in a specific time interval. Furthermore, each model M has a specific instance processing speed $\boldsymbol{v}_M$, defined as the number of instances processable in a specific time interval. For high speed data streams, $\boldsymbol{v}_L >> \boldsymbol{v}_M$ and $\boldsymbol{v}_U >> \boldsymbol{v}_M$. Consequently, not all instances $i_j \in DS_L$ and $i_j \in DS_U$ can be processed by M. As the prediction of $\hat{y_j}$ for all $i_j \in DS_U$ is essential in the application domain, the task is to predict $\hat{y_j}$ as soon as possible after receiving instance i_j from DS_U. This prediction has to be made as good as possible for all $i_j \in DS_U$. Instances without a $\hat{y_j}$ prediction are not allowed in our envisaged usage.

3.2 Approach Specification

To address the given problem setting, we integrate a so-called *controller* into the online process (cf. Figure 1, center). The controller fetches the instances over a specific time interval t_f, which we refer to as *fetching interval*, from the data streams DS_L and DS_U. These instances are defined as X'_{tr} for the instances fetched from the data stream DS_L and X'_{pr} for the instances fetched from DS_U over the time interval t_f. After each fetching interval, the controller uses the instances in X'_{tr} to further train the model M and the instances in X'_{pr} to receive predictions from M. Meanwhile, new instances are collected in the new fetching interval. That way, the controller works as a buffer between the data streams, where the instances can arrive in altering time intervals, and M, where the instances are processed at a constant speed. Furthermore, the controller assures that M is only used by the instances in a time interval of t_f and that the model is then available for the next instances in X'_{tr} and X'_{pr} from the next fetching

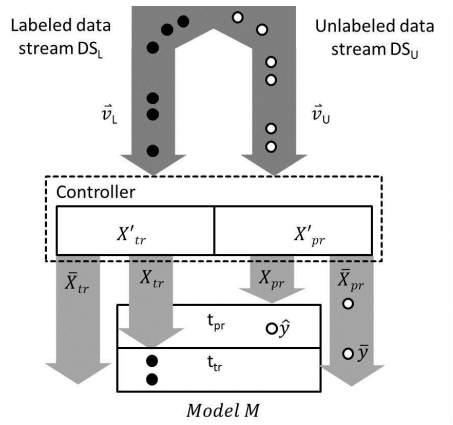


Fig. 1. Illustration of the problem setting. Two data streams DS_U and DS_L provide instances with speed v_L and v_U. The task is to predict $\hat{y}_i$ as soon as possible for each instance from DS_U using model M. To handle very fast data streams, the controller manages the instance processing.

interval. As it is mandatory that all instances $i_j \in X'_{pr}$ receive a value $\hat{y}$, the controller prefers these instances and passes them before the instances in X'_{tr} to the model M. In the case where the data stream speed is constantly higher than the instance processing speed of the model (i.e. $\boldsymbol{v}_U > \boldsymbol{v}_M$), a state is reached where not all instances from DS_U would receive a prediction. Additionally, the model M would not be further improved by the instances from DS_L, because no time is left to process these instances. That is why the time interval t_f is divided into t_{pr} and t_{tr} ($t_{pr} + t_{tr} = t_f$). t_{pr} is the maximal time given to the instances in X'_{pr} to receive a prediction $\hat{y}$ from the model M, and t_{tr} is the maximal time given to the instances in X'_{tr} to improve the model M. As DS_L and DS_U can have different speeds, the amount of instances in X'_{tr} and X'_{pr} can be highly unequal. An equal share of the time would not be suitable. Consequently, t_{pr} is calculated as the proportional number of instances in X'_{pr} over all instances in the controller. Therefore, the time given for the instances in X'_{pr} is

$$t_{pr} = \frac{X'_{pr}}{X'_{pr} + X'_{tr}}.$$

However, as $\boldsymbol{v_M}$ is normally faster for the prediction task than for the training task, there may be the case where the time for the prediction is not fully required, and a prediction for all instances in X'_{pr} is given in t'_{pr} ($t'_{pr} \leq t_{pr}$). Then, the training of the model starts immediately, which gives the instances in X'_{tr} the following maximal time for training:

$$t_{tr} = t_f - t'_{pr}.$$

Such a flexible approach guarantees that the given time is used for both: training and prediction. In the case when $\boldsymbol{v_U}$ ($\boldsymbol{v_L}$) is higher than $\boldsymbol{v_M}$, the controller is

aware that not all instances in X'_{pr} (X'_{tr}) can be processed by M in t_{pr} (t_{tr}). Therefore, only a specific part $X_{pr} \subseteq X'_{pr}$ ($X_{tr} \subseteq X'_{tr}$) can be processed in t_{pr} (t_{tr}), while the rest $\overline{X}_{pr} \subseteq X'_{pr}$ ($\overline{X}_{tr} \subseteq X'_{tr}$) is not provided to the model by the controller. While the controller can discard all instances from $\overline{X}_{tr}$, it is mandatory that also all instances from $\overline{X}_{pr}$ receive a value for the attribute of interest y. As there is no time left in t_{pr}, the model M cannot be applied. To guarantee a prediction, the controller assigns the average value of the attribute of interest over all processed instances $e_i \in DS_L$ ($\overline{y}$) to all instances in $\overline{X}_{pr}$.

4 Experimental Evaluation

This section evaluates the proposed framework on three different data streams, using three different learning algorithms. First, the data stream generation is explained. Second, two alternative frameworks are described, which are used for comparison in the following evaluation. Third, the experimental setup as well as the used learning algorithms are explained. At last, the results are shown and discussed.

4.1 Data Streams

Our approach is evaluated on three different data streams. To simulate the high speed data streams, 1 GB of RAM is filled with instances randomly chosen from each dataset (artificial: 2DimTree[15], real-world: Airline[3], Census[4]) prior to the evaluation process. The data streams DS_L and DS_U are then created by randomly choosing instances from the main memory. Prior storage and fetching of the instances from the main memory is necessary to emulate data streams of sufficiently high speed.

4.2 Alternative Frameworks

There are two straightforward frameworks to handle high speed data streams with a speed higher than the instance processing speed of the model:
The *running-sushi framework (RSF)*, processes only a random subset of the data stream instances (sampling). It is based on the idea to fetch an instance from the data stream DS as soon as the model M is ready to process a new instance. Metaphorically, the data stream can be compared to a conveyor belt in a running sushi bar. There, you always take the next available sushi off the conveyor belt and eat it. As soon as you have finished one piece, you can take the next one. Transferring this idea to the given problem setting, the sushi belt corresponds to the data stream and each single sushi is an instance from DS_U or DS_L. The guest is the model M that processes the instances. The data stream passes the model and each time the model is not processing or has finished processing an instance, the current instance in the stream is selected by the model. Labeled instances are used for training and unlabeled instances receive a prediction from

[3] http://stat-computing.org/dataexpo/2009/
[4] http://archive.ics.uci.edu/ml/datasets/US+Census+Data+%281990%29

the model. As long as the model is processing an instance, all arriving labeled and unlabeled instances from the stream pass.
The *queue framework (QF)* uses an external storage to process all instances by the model M. Each unlabeled instance receives a prediction, and each labeled instance is used to improve the model quality. Instances that cannot be processed by the model immediately are stored in a queue (FIFO principle) for later processing when the resources (time) are available. If the speed of the data stream exceeds the processing speed of the model, the queue extends, and if the data stream speed decreases again, the queue shrinks.

4.3 Experimental Setup

Our framework is compared to the alternative frameworks on all three data streams. To show the flexibility and usability to integrate all kinds of incremental learning algorithms, all three approaches were used with three different incremental learning algorithms: FIMT [16], IMTI-RD [17], IMTI-RA [17][5]. The whole approach as well as the algorithms is written in JAVA. While the FIMT algorithm is a reimplementation based on the published information, the IMTI-RD and IMTI-RA algorithms are original implementations provided by the authors. All three incremental linear model trees were run with default parameters, proposed in the original publication or implementation. All runs were performed on an AMD processor with 2.6 GHz and each JAVA process was given 3800 MB of RAM. The following results are the averaged means of 5 runs using different instance orders in the streams. To motivate our approach, the maximal instance processing speed of each learning algorithm is presented first. Then, each framework is tested using each learning algorithm on all three data streams. The data stream speeds are both set to 1,700,000 instances per second ($\boldsymbol{v}_L = 1,700,000$ instances per second (ips) and $\boldsymbol{v}_U = 1,700,000$ ips). For *PAFAS*, t_f is set to 50 milliseconds. The applicability of the three different frameworks on the given high speed data streams is evaluated based on the number of processed instances, and an analysis of the response time is given. Finally, the prediction accuracy of each framework is compared.

4.4 Results

To test the maximal genuine instance processing speed of each learning algorithm, instances are loaded into the main memory and directly fed to the learning algorithm without any processing system in between. The number of instances that can be used for training (or processed for the prediction respectively) in a second is measured. This can be be interpreted as the maximal data stream speeds ($\boldsymbol{v}_L$ and $\boldsymbol{v}_U$) that can be processed by the algorithm (shown in Table 1). It can be observed that the processing speed of the labeled and unlabeled data streams are very different. Instances from the unlabeled stream (DS_U) can be processed much faster compared to instances from the labeled stream (DS_L).

[5] The frameworks were also tested with a multilayer perceptron neural network, which let to similar results. Due to space limitations, we omitted them here.

Table 1. Maximal data stream processing speed (middle) and the maximal number of collected DS instances i_j until a memory exception takes place for QF (right)

Data stream	Algorithm	Max. $\boldsymbol{v}_L$	Max. $\boldsymbol{v}_U$	Max. i_j using QF (mio)
2DimTree	FIMT	434,852	2,781,893	336
	IMTI-RA	29,551	2,858,142	386
	IMTI-RD	18,612	4,370,217	402
Airline	FIMT	60,909	358,009	253
	IMTI-RA	579	1,424,470	351
	IMTI-RD	45	2,565,789	353
Census	FIMT	52,241	319,892	186
	IMTI-RA	113	801,056	271
	IMTI-RD	5	1,678,321	270

This can be explained by the fact that the learning algorithm is only used on the unlabeled instances X_{pr} for predictions, which is a relatively fast process. In contrast, the labeled instances X_{tr} are used to train, i.e., to improve the model. This process can be, depending on model complexity and data dimensionality, very time-consuming. This becomes evident when comparing the processing speed of the simpler and faster FIMT algorithm to the more complex ones (IMTI-RA and IMTI-RD) over increasing stream complexity (2DimTree to airline to census). This culminates in only 5 instances per second for the IMTI-RD algorithm on the census data stream. However, in real-world applications, the learning algorithms are further embedded in a framework where the instances are fetched from the stream and delivered to them. This framework also consumes CPU time and slows down the algorithm processing speed further.
Memory problems arise for *QF* after a specific number of instances were observed. When using a data stream of 1,700,000 instances per second, which is clearly above every v_L, one can expect that *QF* will run out of memory sooner or later. This depends on the gap between $\boldsymbol{v}_M$ and $\boldsymbol{v}_U + \boldsymbol{v}_L$ and the storage size for the data stream instances. The number of instances that can be collected from the data stream until a memory exception arises is shown in Table 1 for our setting. These numbers suggest that using *QF* for high-speed data streams in real-world applications is not feasible as only few instances can be processed before a system crash. Contrary to *QF*, *RSF* and *PAFAS* can process instances until the framework (including the model) becomes too large for the main memory. As this time span is out of scope, the runs were stopped after fetching 4,294 billion instances (from both streams).

Processed Instances Although an infinite number of instances could be processed by *RSF* and *PAFAS*, it is still important to process as many instances as possible from the data stream. The more instances are included in the training process, the more accurate the predictions should be. And of course in the presented setting every unlabeled instance should also receive a prediction. Thus, the first evaluation addresses the number of processed instances for each framework. *QF* is left out due to its limited usability. Table 2 shows the number of

Table 2. Performance after 4.294 billion instances from the data stream

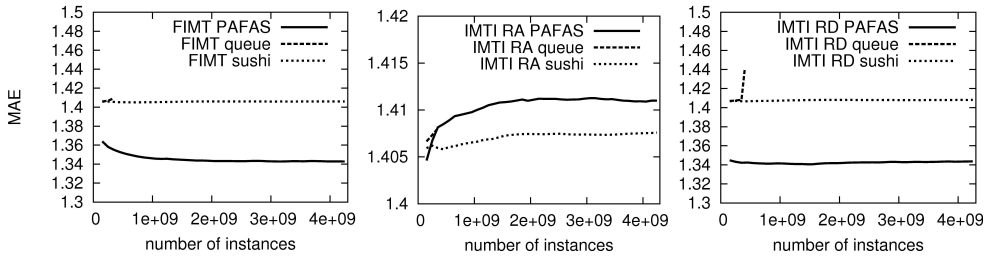
2DimTree					
Algorithm	Framework	#Trained	#Pred. by model	#Pred. with mean	#Missed
FIMT	PAFAS	885,935	110,758,335	2,036,241,665	0
	RSF	9,368,131	9,377,314	0	2,137,622,686
IMTI-RA	PAFAS	622,486	2,195,176	2,144,804,824	0
	RSF	6,852,348	6,859,463	0	2,140,140,537
IMTI-RD	PAFAS	633,274	104,145,808	2,042,854,192	0
	RSF	6,101,393	6,105,633	0	2,140,894,367
Airline					
Algorithm	Framework	#Trained	#Pred. by model	#Pred. with mean	#Missed
FIMT	PAFAS	456,736	57,666,323	2,089,333,677	0
	RSF	7,404,400	7,417,562	0	2,139,582,438
IMTI-RA	PAFAS	382,832	75,899,407	2,071,100,593	0
	RSF	1,228,088	1,230,360	0	2,145,769,640
IMTI-RD	PAFAS	45,780	115,547	2,146,884,452	0
	RSF	37,837	95,054	0	2,146,904,946
Census					
Algorithm	Framework	#Trained	#Pred. by model	#Pred. with mean	#Missed
FIMT	PAFAS	404,284	28,300,148	2,118,699,852	0
	RSF	7,012,848	7,018,232	0	2,139,981,768
IMTI-RA	PAFAS	208,164	3,109,092	2,143,890,908	0
	RSF	193,335	209,721	0	2,146,790,279
IMTI-RD	PAFAS	5,638	164,851	2,146,835,149	0
	RSF	5,432	6,052	0	2,146,993,948

processed instances for each framework after the forced end of each data stream. The number of processed instances for training and testing using *RSF* is nearly equal. This is a consequence of the equal data stream speeds and for that of the equal probabilities to fetch a labeled or unlabeled instance. In contrast, *PAFAS* processes many more prediction instances than training instances. This is done by constantly collecting the instances over the time period t_f and by processing the prediction instances first. The prediction instances are thus preferred over the training instances. More instances are predicted using M, which is reflected in an improved prediction accuracy. Furthermore, the sum of instances that are used for training and prediction by the model is larger for *PAFAS* than for *RSF*. As training time is very costly, *RSF* sacrifices many prediction instances in favor of one training instance. Therefore, the sum of processed instances is much lower than in the *PAFAS* setting.

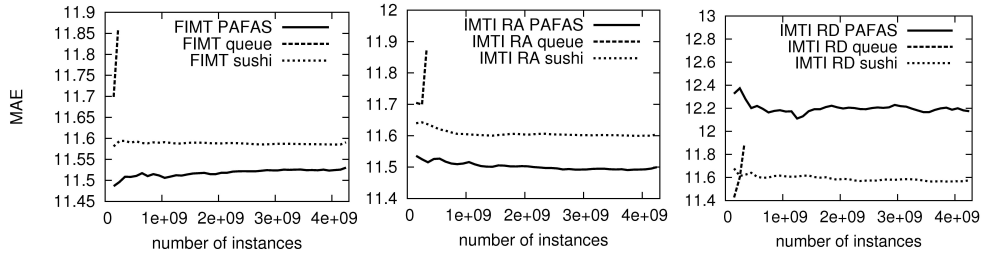
Response Time The next quality criterion is the response time for each framework, i.e. how long it takes for a new unlabeled instance to receive a prediction after appearing in the data stream. The response time of *RSF* is only dependent on the prediction/training time of the model, e.g., a constant response time is observed. In contrast, the *QF* response time increases with the number of instances that are stored in the queue until their prediction. In fact, a linear increase of the

response time can be observed, because a linearly increasing number of instances must be processed before each new instance. This is done in constant time for each instance. Due to lack of space this is not illustrated in this paper. Last, *PAFAS* guarantees a response time not larger than $2 * t_f$ (adjustable parameter) for each unlabeled instance. First, the instances are loaded into the controller, which lasts t_f, and then they are processed in the next time frame, which also lasts t_f. If an instance cannot receive a prediction by the model during this time, the mean target value is assigned to that instance. In both cases, the instance receives a prediction after at most $2 * t_f$.

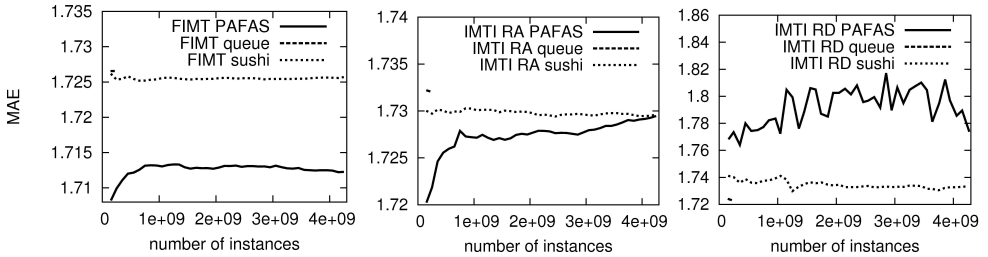
Prediction Accuracy The last quality criterion adresses the prediction accuracy of the instances that have been delivered from the data stream for each framework. However, in *QF* and *RSF* not every unlabeled instance has yet re-



(a) 2DimTree data stream



(b) Airline data stream



(c) Census data stream

Fig. 2. MAE development for all approaches on each data stream.

ceived a prediction, because instances are stuck in the queue (QF) or have been skipped (RSF). In the application domain, leaving out predictions or receiving a delayed prediction may be harmful. Therefore, a penalty is imposed for each instance that is stuck in the queue or was skipped. In both cases, the penalty is set to the average error when using $\overline{y}$ as prediction. This is equal to the non-model prediction made by $PAFAS$ and corresponds to the minimal error that can be made without using a specific model. Of course, the penalty can be adapted to the severity of not predicting an instance, which can then lead to an even worse score. Figure 2 illustrates the MAE for the predictions that have been received from the stream. $PAFAS$ achieves a better MAE in 6 out of 9 cases, because more predictions are made with the model, although it was trained with fewer instances. This may nevertheless be enough for a useful prediction. The cases with an higher error often occur with IMTI-RD on more complex datasets, which could be the result of a temporary overly strong emphasis on the prediction.

5 Conclusion and Future Work

This paper proposes a framework to handle high-speed data streams consisting of labeled and unlabeled instances. It assures that each unlabeled instance receives a prediction in a bounded time interval, while the model is still constantly improved by the labeled instances. Its applicability to three learning algorithms on three data streams has been shown and its performance has been compared to two other approaches. The proposed framework focuses on a single-core implementation yet, as it is a good starting point to develop the approach towards more complex settings. Three interesting adaptations of $PAFAS$ could be addressed in the future. First, the framework could be extended to multicore and distributed systems learning several models. Second, using the advantages of anytime learners, the available training and prediction time could be used more efficiently. For the training instances, sampling variants could be used to choose the most useful ones. On the prediction side, a granularity approach that dynamically decides which depth of the model should be used for the prediction could be tested. There, using sampling is not appropriate, as each instance has to receive a prediction. Third, the time frames could be partitioned into training and prediction times dynamically, as it could be tuned corresponding to the model quality. To obtain a good-quality model, as many instances from X'_{tr} should be provided, which implies that t_{tr} should be as large as possible: $t_{tr} \to t_f$. On the other hand, the user is of course interested in obtaining predictions by the model M for instances X'_{pr}, which implies that t_{pr} should be as large as possible: $t_{pr} \to t_f$. Therefore, it would be interesting to think about the integration of a model-quality dependent time-split decision, possibly using reinforcement learning.

References

1. Stein, L.: The case for cloud computing in genome informatics. Genome Biology **11** (2010) 207

2. Cárdenas, A., Pon, R., Cameron, R.: Management of Streaming Body Sensor Data for Medical Information Systems. In: Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Scienes. (2003) 186–191
3. Chan, D., Krupp, B., Wanchoo, L.: Earth observation system. Technical report, National Aeronautics and Space Administration(NASA) (2010)
4. Moore, G.E.: Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. IEEE Solid-State Circuits Newsletter **20** (2006) 33–35
5. Gaber, M.M.: Advances in data stream mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **2** (2012) 79–85
6. Domingos, P., Hulten, G.: A general method for scaling up machine learning algorithms and its application to clustering. In: In Proceedings of the Eighteenth International Conference on Machine Learning, Morgan Kaufmann (2001) 106–113
7. Wang, F., Yuan, C., Xu, X., van Beek, P.: Supervised and semi-supervised online boosting tree for industrial machine vision application. In: Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data. SensorKDD '11, ACM (2011) 43–51
8. Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: A new model and architecture for data stream management. The VLDB Journal **12** (2003) 120–139
9. Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U., Widom, J.: Stream: The stanford data stream management system. Technical Report 2004-20, Stanford InfoLab (2004)
10. Chi, Y., Wang, H., Yu, P.S.: Loadstar: Load shedding in data stream mining. In: In Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005). (2005) 1302–1305
11. Toivonen, H.: Sampling large databases for association rules. In: Proceedings of the 22th International Conference on Very Large Data Bases (VLDB 1996). (1996) 134–145
12. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '00, ACM (2000) 71–80
13. Gaber, M.: Data stream mining using granularity-based approach. In Abraham, A., Hassanien, A.E., de Leon F. de Carvalho, A., Snel, V., eds.: Foundations of Computational Intelligence. Volume 206 of Studies in Computational Intelligence. Springer Berlin / Heidelberg (2009) 47–66
14. Shieh, J., Keogh, E.: Polishing the right apple: Anytime classification also benefits data streams with constant arrival times. In: Proceedings of the 2010 IEEE International Conference on Data Mining. ICDM '10, IEEE Computer Society (2010) 461–470
15. Vens, C., Blockeel, H.: A simple regression based heuristic for learning model trees. Intelligent Data Analysis **10** (2006) 215–236
16. Ikonomovska, E., Gama, J.: Learning model trees from data streams. In Boulicaut, J.F., Berthold, M., Horvth, T., eds.: Proceedings of the 11th International Discovery Science Conference (DS 2008). Volume 5255 of Lecture Notes in Computer Science., Springer (2008) 52–63
17. Potts, D., Sammut, C.: Incremental learning of linear model trees. Machine Learning **61** (2005) 5–48