IoT Large Scale Learning from Data Streams

Workshop Proceedings

Workshop Organizers:
Moamar Sayed-Mouchaweh, High Engineering School of Mines, Douai, France
Albert Bifet - Telecom-ParisTech, France
Hamid Bouchachia, University of Bournemouth, UK
João Gama, LIAAD-INESC TEC, University of Porto, Portugal
Rita Paula Ribeiro, LIAAD-INESC TEC, University of Porto, Portugal

Contents

Preface

Workshop + Tutorial The volume of data is rapidly increasing due to the development of the technology of information and communication. This data comes mostly in the form of streams. Learning from this ever-growing amount of data requires flexible learning models that self-adapt over time. In addition, these models must take into account many constraints: (pseudo) real-time processing, high-velocity, and dynamic multi-form change such as concept drift and novelty. This workshop welcomes novel research about learning from data streams in evolving environments. It will provide the researchers and participants with a forum for exchanging ideas, presenting recent advances and discussing challenges related to data streams processing. It solicits original work, already completed or in progress. Position papers are also considered. This workshop is combined with a tutorial treating the same topic and will be presented in the same day.

Motivation and focus The volume of data is rapidly increasing due to the development of the technology of information and communication. This data comes mostly in the form of streams. Learning from this ever-growing amount of data requires flexible learning models that self-adapt over time. In addition, these models must take into account many constraints: (pseudo) real-time processing, high-velocity, and dynamic multi-form change such as concept drift and novelty. Consequently, learning from streams of evolving and unbounded data requires developing new algorithms and methods able to learn under the following constraints: -) random access to observations is not feasible or it has high costs, -) memory is small with respect to the size of data, -) data distribution or phenomena generating the data may evolve over time, which is known as concept drift and -) the number of classes may evolve overtime. Therefore, efficient data streams processing requires particular drivers and learning techniques:

- Incremental learning in order to integrate the information carried by each new arriving data;
- Decremental learning in order to forget or unlearn the data samples which are no more useful;
- Novelty detection in order to learn new concepts.

It is worthwhile to emphasize that streams are very often generated by distributed sources, especially with the advent of Internet of Things and therefore processing them centrally may not be efficient especially if the infrastructure is large and complex. Scalable and decentralized learning algorithms are potentially more suitable and efficient.

Aim and scope This workshop welcomes novel research about learning from data streams in evolving environments. It will provide the researchers and participants with a forum for exchanging ideas, presenting recent advances and discussing challenges related to data streams processing. It solicits original work, already completed or in progress. Position papers are also considered. The scope of the workshop covers the following, but not limited to:

- Online and incremental learning
- Online classification, clustering and regression

- Online dimension reduction
- Data drift and shift handling
- Online active and semi-supervised learning
- Online transfer learning
- Adaptive data pre-processing and knowledge discovery
- Applications in
- Monitoring
  - Quality control
  - Fault detection, isolation and prognosis,
  - Internet analytics
  - Decision Support Systems,
  - etc.

A Sliding Window Filter for Time Series Streams

Gordon Lesti[1] and Stephan Spiegel[2]

[1] Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany
gordon.lesti@campus.tu-berlin.de
[2] IBM Research Zurich, Säumerstrasse 4 , 8803 Rüschlikon, Switzerland
tep@zurich.ibm.com

Abstract. The ever increasing number of sensor-equipped devices comes along with a growing need for data analysis techniques that are able to process time series streams in an online fashion. Although many sensor-equipped devices produce never-ending data streams, most real-world applications merely require high-level information about the presence or absence of certain events that correspond to temporal patterns. Since online event detection is usually computational demanding, we propose a sliding window filter that decreases the time/space complexity and, therefore, allows edge computing on devices with only few resources. Our evaluation for online gesture recognition shows that the developed filtering approach does not only reduce the number of expensive dissimilarity comparison, but also maintains high precision.

Keywords: Internet of Things, Time Series Streams, Sliding Window Technique, Online Event Detection, Computational Complexity

1 Introduction

As time goes by things change, and those who understand change can adapt accordingly. This basic principle is also reflected in today's digital society, where sensor-equipped devices measure our environment and online algorithms process the generated data streams in quasi real-time to inform humans or cognitive systems about relevant trends and events that impact decision making.

Depending on the domain researchers either speak about events, patterns, or scenes that they aim to detect or recognize in time series, sensor, or data streams. Applications range from event detection for smart home control [17] over frequent pattern mining for engine optimization [16] to scene detection for video content [1] and gesture recognition for human-computer interaction [11].

Commonly online algorithms for data streams employ the popular sliding window technique [8], which observes the most recent sensor measurements and moves along the time axis as new measurements arrive. Usually each window is examined for a set of predefined events, which requires the comparison of the current time series segment and all preliminary learned instances of the relevant temporal patterns. In general, the pairwise dissimilarity comparisons of temporal patterns are performed by time series distance measures [18].

The time and space complexity of the sliding window technique increases with decreasing step size as well as growing window size, measurement frequency, and number of preliminary learned instances. High computational demand and memory usage is especially problematic for embedded systems with only few resources [9,19], which applies to the greatest part of sensor-equipped devices within the typical Internet of Things (IoT) scenario.

Our aim is to reduce the number of computational expensive dissimilarity comparisons that are required by the sliding window technique. To this end we propose a sliding window filter [10], which is able to decide whether the current window should be passed to a time series classifier or not. Although the filter could be considered as a binary classifier itself, it merely employs statistical measures with linear complexity and, thereby, avoids using computationally more expensive dissimilarity comparisons in many cases. Our approach to mitigate the computational complexity of event detection in data streams is different from other techniques in that we refrain from accelerating time series distance measures [13,15] or reducing dataset numerosity [20].

We demonstrate the practical use of our proposed sliding window filter for gesture recognition in continuous streams of accelleration data [10,11], where a great amount of the necessary but expensive Dynamic Time Warping (DTW) distance calculations [7] is replaced by less demanding statistical measures, such as the complexity estimate [2] or sample variance [3]. Our experimental results show that the number of DTW distance calculations can be cut in half, while still maintaining the same high gesture recognition performance.

The rest of the paper is structured as follows. Chapter 2 introduces background and notation. Chapter 3 and 4 introduce and evaluate our proposed sliding window filter. We conclude with future work in Chapter 5.

2 Background and Notation

This section gives more background on the sliding window technique [8], DTW distance measure [7], and time series normalization [4], which are fundamental building blocks of our conducted online gesture recognition study [10]. Table 1 introduces the notation that we use for formal problem description.

Symbol	Description
Q	a time series of size n with $Q = (q_1, q_2, \dots, q_i, \dots, q_n)$
$Q[i, j]$	a subsequence time series of Q with $Q[i, j] = (q_i, q_{i+1}, \dots, q_j)$
t	the current time
μ	the mean of a time series Q
σ	the standard deviation of a time series Q
η, z	two different time series normalizations

Table 1. Notation used for formal problem description.

2.1 Sliding Window Technique

Given a continuous time series stream Q, the sliding window technique examines the w most recent data points and moves s steps along the time axis as new measurements arrive, where w and s are referred to as window and step size. This technique has the advantage that it does not need to store the never-ending stream of data, but it also implies that measurements can only be considered for further data analysis as long as they are located within the current window.

In most applications, each window is passed to a data processing unit, which performs some kind of time series classification, clustering, or anomaly detection. For example in online gesture recognition [10], one can employ a nearest neighbor classifier, which compares each window to a training set of preliminary learned time series instances. In case that the current window $Q[t-w, t]$ is similar to one of the known gestures, where similar means that the time series distance falls below a certain threshold, a corresponding action can be triggered. A popular distance measure for gestures [11] and other warped time series is described in the following subsection.

2.2 Dynamic Time Warping

Dynamic Time Warping (DTW) is a widely used and robust distance measure for time series, *allowing similar shapes to match even if they are out of phase in the time axis* [7]. Traditionally DTW computes a full distance matrix to find an optimal warping path, where possible nonlinear alignments between a pair of time series include matches of early time points of the first sequence with late time points of the second sequence. To prevent pathological alignments, the size of the warping window can be constraint, for instance, by the Sakoe-Chiba band [12] or the Itakura parallelogram [6]. Figure 1 illustrates the DTW distance measure using a Sakoe-Chiba band of 10%, where the percentage of the warping window refers to the length of the compared time series.

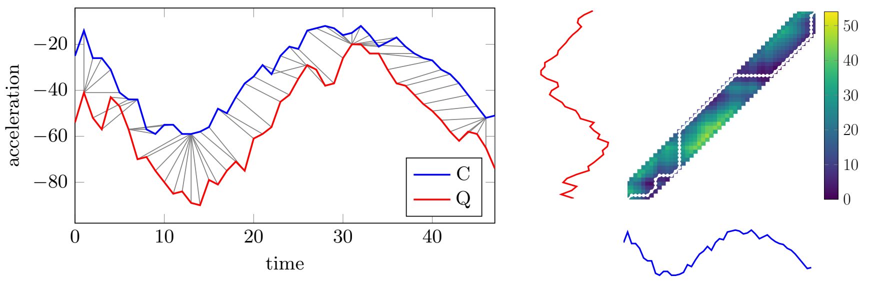


Fig. 1. Time series Q and C compared by DTW with a Sakoe-Chiba band of 10% time series length. The left plot illustrates the nonlinear alignment between the two sequences and the right plot shows the optimal warping path within the specified band.

2.3 Time Series Normalization

Literature on time series mining [5,18] suggests to normalize all (sub-)sequences before measuring their pair-wise dissimilarity by means of a distance measure. There are multiple ways to normalize time series, where two common techniques [4] are compared in this study.

Given is a time series $Q = (q_1, \ldots, q_n)$ of length n, we can compute its mean μ and standard deviation σ as followed:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} q_i \qquad\qquad \sigma = \frac{1}{n}\sum_{i=1}^{n}(q_i - \mu)^2$$

Having defined the mean μ and standard deviation σ, we can normalize each data point q_i (with $1 \leq i \leq n$) of a time series $Q = (q_1, \ldots, q_n)$ in one of the two following ways [4]:

$$\eta(q_i) = q_i - \mu \tag{1}$$

$$z(q_i) = \frac{q_i - \mu}{\sigma} \tag{2}$$

Equation 2 is commonly known as the Z-score. For the sake of simplicity we refer to η and z normalization [4] for the rest of the paper.

3 Filtering Approach

This section does not only explain the concept of our proposed filtering approach, but also describes how to integrate our filter into the well-known and widely-used sliding window technique, as shown in Figure 2.

In general, the sliding window filter considers the most recent measurements in a data stream. The considered measurements are usually passed to a classifier, which aims at categorizing the current time series subsequence. In case that the current subsequence was assigned to a known category or class, a corresponding action is triggered and the next non-overlapping window, w steps along the time axis, is examined. If the current subsequence just contains noise and no category was assigned, the next overlapping window, s steps along the arrow of time, is processed. The main limitation of this traditional sliding window technique is its computational complexity, which increases with growing window size, shrinking step size, higher sample rate, and larger training set.

For instance, given a data stream of length l=10090, a window size of w=100, and a step size of s=10, we need to classify $(l - (w - s))/s = 1000$ windows. Moreover, assuming 20 training time series, classifying 1000 windows by means of the nearest neighbor approach requires exactly $20 * 1000 = 20K$ dissimilarity comparisons. In case that we employ unconstrained DTW as time series distance measure, we need to compute $20K$ full warping matrices, each of them containing $w * w = 10K$ cells, resulting in a total amount of $200M$ distance operations.

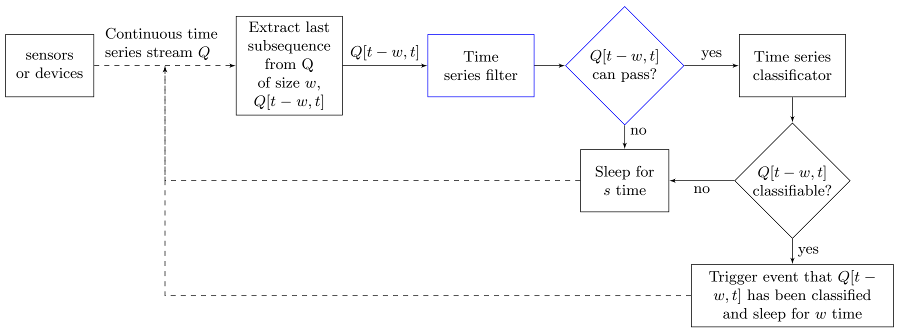

sensors or devices
Continuous time series stream Q
Extract last subsequence from Q of size w, $Q[t-w,t]$
$Q[t-w,t]$
Time series filter
$Q[t-w,t]$ can pass?
yes
no
Time series classificator
Sleep for s time
no
$Q[t-w,t]$ classifiable?
yes
Trigger event that $Q[t-w,t]$ has been classified and sleep for w time

Fig. 2. Flowchart of sliding window technique with filter, highlighted in blue. The current time is denoted by t. Window and step size are denoted by w and s respectively.

In order to reduce the large number of computational expensive dissimilarity comparisons, we propose to employ a sliding window filter, which is capable of separating signal from noise, only passing promising time series subsequences to the classifier. In that sense, the proposed filter can also be considered as a binary classifier, which prunes windows that are likely to be noise and forwards subsequences that exhibit similar features as the training time series. Extracting characteristic time series features that can be used as a filter criterion should ideally exhibit linear complexity, because we aim at replacing more expensive dissimilarity comparisons. Suitable filter candidates include statistical measures, such as the sample variance [3] and length normalized complexity estimate [2], explained in more detail below.

Given is a time series $Q = (q_1, \ldots, q_n)$ with length n, we can define the sample variance (VAR) and length normalized complexity estimate ($LNCE$) as follows:

$$VAR(Q) = \frac{1}{n} \sum_{i=1}^{n} (q_i - \mu)^2$$

$$LNCE(Q) = \frac{1}{n-1} \sqrt[2]{\sum_{i=1}^{n-1} (q_i - q_{i+1})^2}$$

Having defined the above statistical measures, we are able to compute the VAR and $LNCE$ for all training time series and, subsequently, use the resulting range of statistical values to learn an appropriate filter interval. During testing, each window that exhibits a measured value within the learned interval is passed to the classifier or pruned otherwise. In order to avoid excessive pruning of relevant windows, we further more introduce a multiplication factor, which allows us to expend the interval boundaries by a certain percentage.

In general, we aim at designing a filter with high precision and recall. In our case, precision is the ratio between the number of relevant windows that were passed to the classifier (true positives) and the number of all windows that were passed to the classifier (true positives and false positives). Consequently, recall is the ratio between the number of relevant windows that were passed to the classifier (true positives) and the number of all relevant windows (true positives and false negatives). An exhaustive evaluation of our proposed sliding window filter in dependence of all model parameters is presented in the next section.

4 Evaluation

This chapter describes the data aggregation in Section 4.1, data preparation in Section 4.2, experimental setup in Section Section 4.3, and used performance measures in Section 4.4, before presenting our results in Section 4.5.

4.1 Data Aggregation

We employed a Wii Remote^{TM} Plus controller to record different gestures for multiple users. Each user performed 8 gestures, first in a controlled environment to record clean training samples and afterwards in noisy environment to record a test time series stream, which includes all predetermined gestures as well as acceleration data that corresponds to other physical activities. All records are available for download on our project website [10]. A sample record containing both training and test gestures is illustrated in Figure 3.

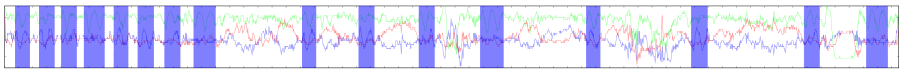


Fig. 3. An sample record of 98 seconds length, containing 8 training gestures at the very beginning, directly followed by a test time series stream that comprises the same 8 gestures mixed with acceleration data of various physical activities. Gestures are highlighted by blue rectangles and are also marked in the recorded data. Note that the length of gestures can vary for training and test set as well as for different records.

4.2 Data Preparation

All of our data records are resampled and quantized before further analysis. In general, dimensionality and cardinality reduction of time series is performed to ease and accelerate data processing by providing a more compact representation of equidistant measurements [11].

Resampling: The recorded acceleration data was resampled by means of the moving average technique, using a window size of 50 ms and step size of 30 ms.

Acceleration data (a) in $\frac{dm}{s^2}$	**Converted value**
$a > 200$	16
$100 < a < 200$	11 to 15 (five levels linearly)
$0 < a < 100$	1 to 10 (ten levels linearly)
$a = 0$	0
$-100 < a < 0$	-1 to - 10 (ten levels linearly)
$-200 < a < -100$	-11 to - 15 (five levels linearly)
$a < -200$	-16

Table 2. Conversion of recorded acceleration data from $\frac{dm}{s^2}$ scale to integer values.

Quantization: The resampled records were then converted into time series with integer values between -16 and 16, such as suggested in related work [11] and summarized in table 2.

4.3 Experiment

The proposed sliding window filter has several model parameters that need to be carefully tuned in order to achieve optimal performance. Depending on the application domain we need to select an appropriate window and step size, time series normalization, dissimilarity threshold, and filter criterion. In the following we describe all parameter settings that were assessed in our empirical study:

– The **window size** determines the number of most recent measurements contained in the examined time series subsequences. We tested four different sizes that were learned from the training gesture, including **min**, **max**, and **avg** length as well as the **mid**-point of the range.

– The **step size** defines the gap between consecutive time series windows. As default setting we use one tenth of the window size.

– For online gesture recognition we employ the nearest neighbor classifier in combination with the DTW distance, where we evaluate 34 different Sakoe-Chiba **band** sizes, ranging from 1 % to 100 %. Prior to pair-wise comparing sliding windows and training gestures, the corresponding time series should be normalized. We evaluate η, z, and no **normalization**.

– The dissimilarity **threshold** defines the time series distance at which a sliding window and a training gesture are considered to belong to the same class. We determine the threshold for an individual class by measuring the distances between all samples of that particular class and all instances of other classes. In our empirical study we evaluate the threshold influence for: (i) one half of the minimum distance - **HMinD**, (ii) one half of the average distance - **HAvgD**, and (iii) one half of the midpoint distance - **HMidD**.

– The **filter criterion** is an essential part of our proposed approach. In our empirical study we evaluate the performance of the two filter criteria, namely the sample variance **VAR** and the length normalized complexity estimate **LNCE** of a time series. Both filters are tested with different factors that increase the size of the filter interval from 100 % to 300 %.

Figure 4 visualizes the online gesture recognition results for a sample time series stream processed by our proposed sliding window filter, after selecting the above described model parameters with help of the recorded training gestures.

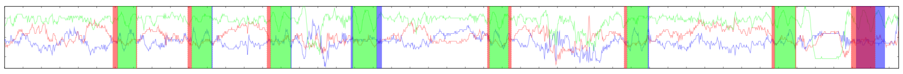


Fig. 4. Visualized results of online gesture recognition for a sample time series stream. We highlight true positives in green, false positives in red, false negatives in blue, and true negatives in transparent. Although we see short false detection intervals before or after true positives, seven out of eight gestures were assigned to the correct class label.

4.4 Performance Measures

Since our proposed sliding window filter is tested on time series streams that contain various different gestures, we need to treat the described online gesture recognition challenge as multi-class problem. Common performance measures for multi-class problems are $Precision_\mu$, $Recall_\mu$ and $F_\beta score_\mu$ [14]:

$$Precision_\mu = \sum_{i=1}^{l} tp_i \Big/ \sum_{i=1}^{l} (tp_i + fp_i)$$

$$Recall_\mu = \sum_{i=1}^{l} tp_i \Big/ \sum_{i=1}^{l} (tp_i + fn_i)$$

$$F_\beta score_\mu = (\beta^2 + 1) Precision_\mu Recall_\mu \Big/ \beta^2 Precision_\mu + Recall_\mu$$

where β is usually set to one and l denotes the number of classes that require separate computation of true positives (tp), false positives (fp), and false negatives (fn). These multi-class performance measures allow us to compare and rank the results for different parameter settings. For our evaluation we employ the $F_1 score_\mu$, which weights $Precision_\mu$ and $Recall_\mu$ equally.

4.5 Results

In order to evaluate the influence of all model parameters that were described in Section 4.3, we performed a total number of 28152 experiments. Figure 5(a) illustrates the $Precision_{\mu}$ and $Recall_{\mu}$ values for all test runs. A top performance of around 0.7384 $F_{1}score_{\mu}$ was achieved by parameter configurations that used η time series normalization, DTW with a Sakoe-Chiba band of about 18 % time series length, *mid* window size, and *HAvgD* for threshold determination.

Given the best parameter configuration, we investigated the influence of the individual parameters by changing only one at a time and fixing the others, see Figure 5(b,c,d). As shown in Figure 5(e), we also evaluated the performance with VAR, $LNCE$, and no filter. The best results for each individual gesture is shown in Figure Figure 5(f). Further tests on the applicability of the sliding window filter as well as our interpretation of the results are presented below.

Normalization: The influence of the time series normalization is illustrated in Figure 5(b). We compare η, z, and no normalization, with *mid* window size and *HAvgD* dissimilarity threshold. The best $F_{1}score_{\mu}$ was achieved by means of the η normalization, which corresponds to the data points shown in the magnifying glass. The point cloud in the lower left corner of plot 5(b) are parameter settings with rather small warping band.

Warping Band: The influence of the Sakoe-Chiba band in combination with the DTW distance is shown below in Figure 6. For this experiment we selected only the dominating parameter settings, with η normalization, *mid* window size, and and *HAvgD* dissimilarity threshold. The best $F_{1}score_{\mu}$ was achieved with a band with of 18 % time series length.

Dissimilarity Threshold: We evaluate three different ways of determining a dissimilarity threshold, namely *HMinD*, *HAvgD*, and *HMidD*. For our comparison in Figure 5(c), we used η normalization, *mid* window size, and a warping band of 18 % time series length. The best $F_{1}score_{\mu}$ was achieved by means of *HAvgD*, shortly followed by the *HMidD* approach. Comparatively high $Precision_{\mu}$ values were given by *HMinD* threshold.

Window Size: The influence of the window size determination approach is shown in Figure 5(d). We compare *min*, *max*, *avg*, and *mid* window size, with η normalization, *HAvgD* dissimilarity threshold, and a warping and of 18 % time series length. The highest $Precision_{\mu}$, $Recall_{\mu}$ and $F_{1}score_{\mu}$ was achieved by the *mid* window size, shortly followed by the *avg* window size. Figure 5(d) furthermore suggests to refrain from using *max* and *min* window size determination.

Filtering Approach: Given the optimal parameter setting that was determined in the previous experiments, we are now in the position to assess the influence of the filtering approach. Figure 5(e) shows the performance with VAR, $LNCE$, and no filter. Twenty simulations are reaching a $F_{1}score_{\mu}$ value greater or equal to 0.7. Interestingly, top performance was achieved with and without filter. This lead is to the question of computational complexity, which is answered in following paragraph.

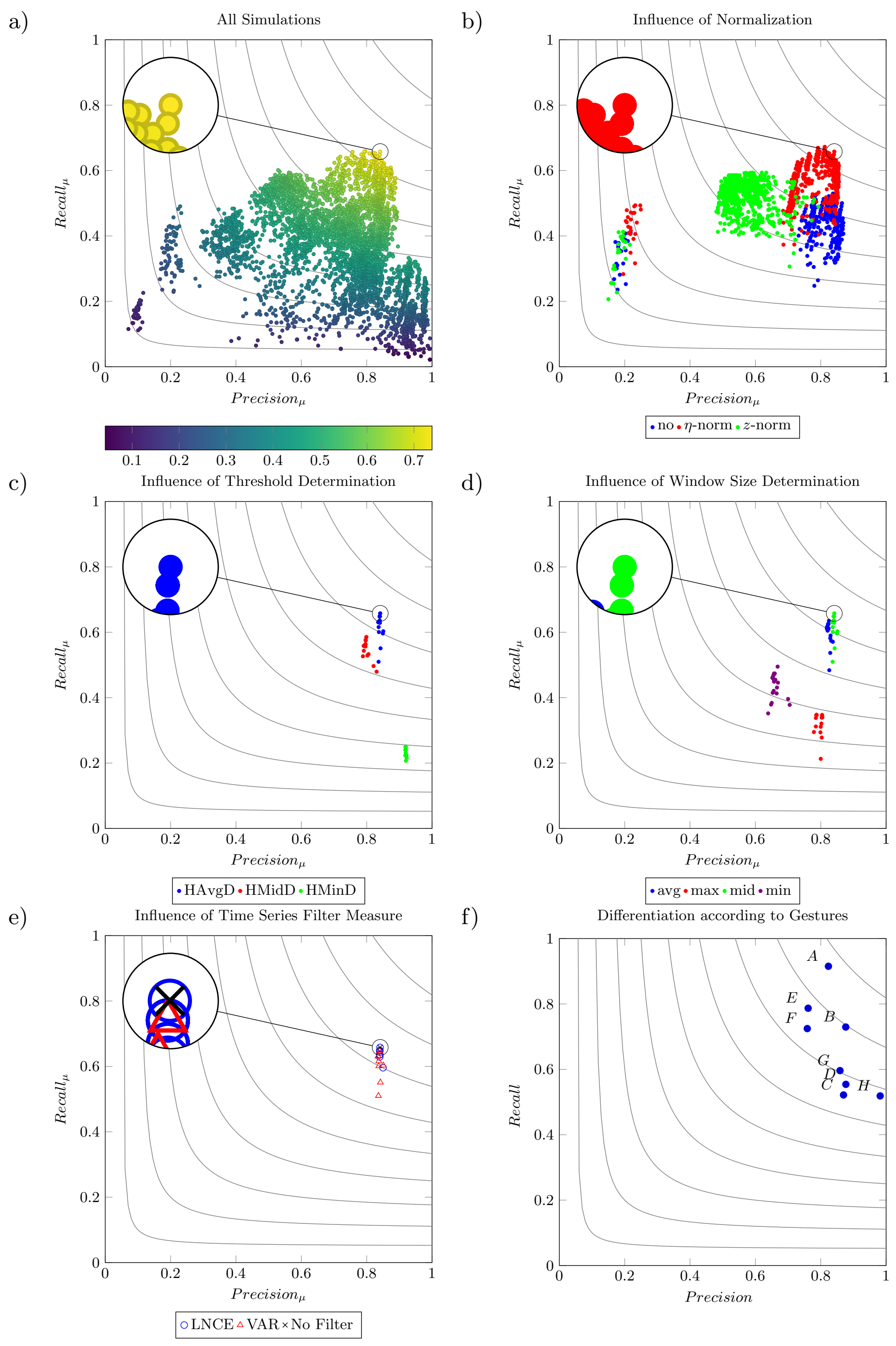


Fig. 5. $Precision_\mu$ and $Recall_\mu$ plots illustrating the performance influence of the individual model parameters. The magnifying glass is focusing on the results with the highest F_1score_μ. Gray lines indicate the F_1score_μ distribution in $\frac{1}{10}$ steps.

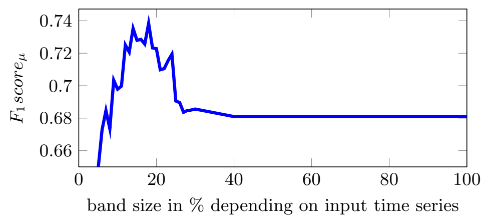


Fig. 6. Performance with varying Sakoe-Chiba band width.

Filter Interval: The filter interval does not only influence the resulting $F_1 score_\mu$, but also the amount of time series dissimilarity comparisons. Figure 7 illustrates the influence of the interval size in respect to performance and computational demand. With an appropriate filter interval of about 200 % we are able to reduce the number of dissimilarity comparisons by one half, while still achieving relatively high performance values.

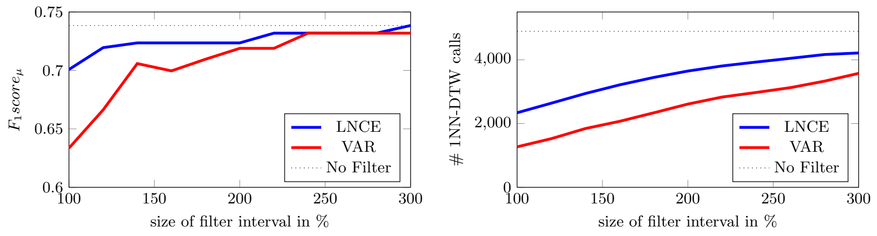


Fig. 7. Influence of filter interval on the $F_1 score_\mu$ (left) and the amount of 1NN-DTW dissimilarity comparisons (right). An optimal tradeoff between performance and computational demand is achieved at approximately 150 % to 200 % filter interval.

Individual Gestures Finally, we have tested the performance for each of the examined gestures separately. Figure 5(e) shows the best *Precision* and *Recall* values that were achieved for each gesture. The results demonstrate that some gestures are easier to recognize than others.

5 Conclusion and Future Work

In this work we have proposed a novel sliding window filter for more efficient event detection in time series streams, which replaces computational expensive dissimilarity comparisons by less demanding statistical measures. Furthermore, we have demonstrated that the developed filter is able to recognize gestures in continuous streams of acceleration data with high accuracy, while cutting the number of distance calculations in half.

Possible applications do not only include event detection on mobile device with few hardware resources, but also distributed sensor networks with limited bandwidth that communicate high level information instead of transferring raw data. In future work we plan to investigate a larger variety of statistical measures that exhibit favorable filtering properties for online gesture recognition as well as data streams found in other domains.

References

1. E. Acar, S. Spiegel, and S. Albayrak. MediaEval 2011 Affect Task: Violent Scene Detection combining audio and visual Features with SVM. CEUR Workshop, 2011.
2. G. Batista, X. Wang, and E. J. Keogh. A complexity-invariant distance measure for time series. ICDM, 2011.
3. T. F. Chan, G. H. Golub, and R. J. LeVeque. Algorithms for computing the sample variance: Analysis and recommendations. The American Statistician,1983.
4. G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule Discovery from Time Series. KDD, 1998.
5. H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. VLDB Endowment, 2008.
6. F. Itakura. Minimum prediction residual principle applied to speech recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975.
7. E. Keogh. Exact indexing of dynamic time warping. VLDB Endowment, 2002.
8. E. Keogh, S. Chu, D. Hart, and M. Pazzani. Segmenting time series: A survey and novel approach. Data mining in Time Series Databases, 2004.
9. S. Kratz, M. Rohs, K. Wolf, J. Mueller, M. Wilhelm, C. Johansson, J. Tholander, J. Laaksolahti. Body, movement, gesture and tactility in interaction with mobile devices. MobileHCI, 2011.
10. G. Lesti and S. Spiegel. The sliding window filter for time series streams website. https://gordonlesti.com/a-sliding-window-filter-for-time-series-streams/
11. J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uWave: Accelerometer-based personalized gesture recognition and its applications. PerCom, 2009.
12. H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. Transactions on acoustics, speech, and signal processing, 1978.
13. D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul. Accelerating dynamic time warping subsequence search with GPUs and FPGAs. ICDM, 2010.
14. M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. Information Processing and Management, 2009.
15. S. Spiegel, B.-J. Jain, and S. Albayrak. Fast time series classification under lucky time warping distance. SAC, 2014.
16. S. Spiegel. Discovery of driving behavior patterns. Smart Information Systems - Advances in Computer Vision and Pattern Recognition, 2015.
17. S. Spiegel. Optimization of in-house energy demand. Smart Information Systems - Advances in Computer Vision and Pattern Recognition, 2015.
18. S. Spiegel. Time series distance measures: segmentation, classification and clustering of temporal data. TU Berlin, 2015.
19. M. Wilhelm , D. Krakowczyk, F. Trollmann, S. Albayrak. eRing: Multiple Finger Gesture Recognition with one Ring Using an Electric Field. iWOAR, 2015.
20. X. Xi, E. Keogh, C. Shelton, L. Wei, C. A. Ratanamahatana. Fast time series classification using numerosity reduction. ICML, 2006.

Multi-model Optimization with Discounted Reward and Budget Constraint

Jixuan Shi and Mei Chen*

Big Data Science Lab, Beijing Neucloud Technology Co. Ltd,China,
shijixuan@neucloud.cn, mei.chen.web@gmail.com

Abstract. Multiple arm bandit algorithm is widely used in gaming, gambling, policy generation, and artificial intelligence projects and gets more attention recently. In this paper, we explore non-stationary reward MAB problem with limited query budget. An upper confidence bound (UCB) based algorithm for the discounted MAB budget finite problem, which uses reward-cost ratio instead of arm rewards in discount empirical average. In order to estimate the instantaneous expected reward-cost ratio, the DUCB-BF policy averages past rewards with a discount factor giving more weight to recent observations. Theoretical regret bound is established with proof to be over-performed than other MAB algorithms. A real application on maintenance recovery models refinement is explored. Results comparison on 4 different MAB algorithms and DUCB-BF algorithm yields lowest regret as expected.

1 Introduction

Multiple model refinement with limited budget is a common and essential challenge in many decision support and learning applications, such as recovery models for different maintenance action, risk assessment models according to different risk levels, or hypothesis learning for multi-hypotheses test. The model learning and refinement (reduce uncertainty) was addressed in a passive way mostly with individual model separately. However, in many case, it is important and critical to reduce overall uncertainties (refine models) from a systematic view. Apart from fundamental needs of model learning, we may face challenges due to sparse training data (e.g. data is rare for high risk model), budget constraint with different action costs (e.g. high cost maintenance action data), and complicated uncertainty models.

In this paper, we model the integrated multiple model refinement process as a sequential decision making problem and apply multi-armed bandit algorithm for optimization. The multi-armed bandit algorithm has been studied extensively in the literature due to its capability of sequential decision support in an uncertainty environment [1], [2], [5], [12]; and widely applied in real applications, including web search [9], online advertising [8], recommendation [11], multi-agent systems [10], queuing & scheduling [3], [4], and so on.

MAB algorithm models allocation resource under uncertainty to maximize the reward, where typical assumption is a bandit with K independent arms with stationary reward. At each time step t, the agent chooses one arm and receives a reward accordingly. In the stationary case, the distribution of the rewards are initially unknown, but are assumed to be unchanged in whole process. The agent aims at minimizing the expected regret over T rounds, which is defined as the expectation of the difference between the total reward obtained by playing the best arm and the total reward obtained by using the algorithm.

Though the stationary formulation of the MAB problem allows to address exploration versus exploitation challenges in a intuitive and elegant way, it may fail to be adequate to model an evolving environment where the reward distributions undergo changes in time. As an example, in the famous mini-game: Gold Miner Go, a player wishes to opportunistically dig as many gold mines as possible; the arms are the different digging direction the player can choose; the reward is the gain of each approach, whose distribution is unknown to the player. The reward is obviously not stationary in this case.

To model such situations, we need to consider non-stationary MAB problems, where distributions of rewards may change in time. Moreover, when budget is constrained, which means we may need to select the 'arm' more carefully in each step. Where the budget can be expensive expert tickets or limited game time.

This paper is organized as follows. Detailed problem setup and corresponding algorithm are explored in section 2. Expected regret over finite rounds is estimated in section 3 with low bound of pulling time T. We use a multiple recovery model refinement problem in schedule maintenance as example in experiment and algorithm implementation. The results are shown in section 4 with comparison between several state-of-art MAB algorithms, with conclusion in section 5.

2 Problem and Algorithm

Note that the major difference between discounted and stationary MAB problems is that the rewards for each arm are modeled by a sequence of independent random variables from potentially different distribution (unknown to the user) which may vary across time. Given this, the operators goal is to maximize the sum of rewards it earns from pulling the arms of the system but not sticking on some arms like stationary MAB does.

When considering the cost on action 'k' versus its reward μ_k, we believe that we can further optimize the selective query by using the reward rate $\frac{\mu_k}{c_k}$ [13], [14]. When considering the discounted (non-stationary) reward on each arm (the model uncertainty gets small when getting more samples), proper adjustment on averaged reward $\hat{\mu}_k(t)$ should be considered [6], [7], [15].

This is a typical optimization problem applied in many scenarios:

- model uncertainty reduction by asking for more data points;
- hierarchical clustering to reduce the node impurity by querying which node;
- hierarchical active learning with steaming data;
- trigger time for online learning (query) to reduce classification/prediction error.

The general conditions are:

– budget constraint and (different) fixed cost arms;
– discounted reward, which can be linear or nonlinear reduction.

all the models in the project to minimize his/her total
To solve this problem, we design an upper confidence bound (UCB) based algorithm for the D-MAB-BF problem, as shown in the following.

Algorithm 1 The DUCB-BF Algorithm

```
1: Initialization: Pull each arm i once in the first K steps, set t = K.
2: while $\sum_{i=1}^{K} c_i N_t(1,i) < B$ do
3:     Set $t = t + 1$.
4:     Calculate the index $D_{i,t}$ of each arm i as $D_{i,t} = \bar{X}_t(\gamma, i) + c_t(\gamma, i)$
5:     Pull the arm at with the largest index: $I_t = \arg\max_{1\le i\le K} D_{i,t}$
6: end while
7: return $t_B = t - 1$.
```

The definitions of $\bar{X}_t(\gamma, i)$ is given as

$$\bar{X}_t(\gamma, i) = \frac{1}{N_t(\gamma, i)} \sum_{s=1}^{t} \gamma^{t-s} X_s(i) I(A_s = i)/c_i,$$

$$N_t(\gamma, i) = \sum_{s=1}^{t} \gamma^{t-s} I(A_s = i).$$

The definition of $c_t(\gamma, i)$ is defined as
$c_t(\gamma, i)=2\sqrt{\frac{\xi \log n_t(\gamma)}{N_t(\gamma,i)}}$, $n_t(\gamma) = \sum_{i=1}^{K} N_t(\gamma, i)$. I is the indicator function in previous analysis.

DUCB-BF algorithm uses reward-cost ratio instead of arm rewards in discount empirical average. In order to estimate the instantaneous expected reward-cost ratio, the DUCB-BF policy averages past rewards with a discount factor giving more weight to recent observations.

3 Regret Analysis

To formulate the statement about the regret analysis here, let $\mu_t(i)$ be the expected reward gained when are i is pulled at time t, $\mu_t^* = \max_i \mu_t(i)$ be the maximal reward at time t and $\varDelta_t(i) = \mu_t^* - \mu_t(i)$.

The total expected regret of the operator is:

$$\begin{aligned}\mathrm{E}[R_T] &= \mathrm{E}\left[\sum_{s=1}^{T}\mu_s^* - \sum_{i=1}^{K}\sum_{s=1}^{T}\mu_s(i)I(A_s=i)\right]\\ &= \mathrm{E}\left[\sum_{i=1}^{K}\sum_{s=1}^{T}\Delta_t(i)I(A_s=i)\right]\end{aligned}$$

In stationary MAB problems, $\mu_t(i) = \mu_i$ remains the same for all times $t \in \{1,\dots,T\}$, and $\mu_t^* = \mu^*$, $\Delta_t(i) = \Delta_i$. The total expected regret is simplified to:

$$\begin{aligned}\mathrm{E}[R_T] &= \mathrm{E}\left[\sum_{i=1}^{K}\sum_{s=1}^{T}\Delta_t(i)I(A_s=i)\right]\\ &= \sum_{i=1}^{K}\Delta_i\mathrm{E}\left[\sum_{s=1}^{T}I(A_s=i)\right] = \sum_{i=1}^{K}\Delta_i\mathrm{E}[N_T(1,i)]\end{aligned}$$

Let $\widetilde{N}_T(i)$ denote the number of times arm i played when it was(is) not the best arm during T rounds:

$$\widetilde{N}_T(i) = \sum_{s=1}^{T} I(A_s = i \neq i_s^*).$$

$\widetilde{N}_T(i)$ represents how many suboptimal choice the algorithm had made.

In non-stationary MAB problems, each $\mu_t(i)$ is unpredictable. No theoretical expression on μ_t^* when the reward is reducing regularly. For simplification, we can give a fix value on δ_i and then we only need to analyze $\widetilde{N}_T(i)$.

In some situations, the expected reward gained from arm i is changing in certain way. We consider a particular non-stationary case where the distribution of the rewards is reducing regularly. The reducing rate of arm i is denoted by ε_i. We also assume that the variance of the reward gain is also reducing. The reward gain from arm i is a sequence of independent random variables denoted by $\varepsilon_i^{t-1}X_t(i)$, which $X_1(i), X_2(i), \dots$ be a sequence of independent random variables. played, when this arm is suboptimal.

Theorem 1 *Let $\xi > 1/2$ and $\gamma \in (0,1)$. For any arm $i \in \{1,\dots,K\}$*

$$E\left[\widetilde{N}_T(i)\right] \leq \frac{16\xi \log n_T(\gamma)\gamma^{-(K+1)}}{(\Delta\mu_T(i))^2} + 2\left[\frac{-\log(1-\gamma)}{\log(1+4\sqrt{1-\xi/2})}\right]\frac{(T-K)(1-\gamma)}{1-\gamma^{K+1}}$$

where $\Delta\mu_T(i) = \min\{s \in \{1,\dots,T\}, i \neq i_s^*, \mu_t(*) - \mu_t(i)\}$

Proof The proof is adapted from [7]. There are however two main differences. First, because the expect reward is reducing regularly, there is no break point. In addition, the expect reward is keeping changing during the process, we cannot regard the process as piecewise stationary process. The proof is in 3 steps:

Step 1 We upper-bound the number of times the suboptimal arm i is played as follows:

$$\begin{aligned}
\widetilde{N}_T(i) &= \sum_{s=1}^{T} I(A_s = i \neq i_s^*) \\
&\leq 1 + \sum_{s=K+1}^{T} I(A_s = i \neq i_s^*) \\
&\leq 1 + \sum_{s=K+1}^{T} I(A_s = i \neq i_s^*, N_s(\gamma, i) < L) + \sum_{s=K+1}^{T} I(A_s = i \neq i_s^*, N_s(\gamma, i) \geq L)
\end{aligned} \tag{1}$$

Step 2 We observe that

$$N_t(\gamma, i) = \sum_{s=1}^{t} \gamma^{t-s} I(A_s = i) > \gamma^t \sum_{s=1}^{t} I(A_s = i) = \gamma^t N_t(1, i)$$

The upper-bound of the second sum in the equation(1) is

$$\begin{aligned}
\sum_{s=K+1}^{T} I(A_s = i \neq i_s^*, N_s(\gamma, i) < L) &\leq \sum_{s=K+1}^{T} I(A_s = i, \gamma^s N_s(1, i) < L) \\
&\leq \gamma^{-(K+1)} L - 1
\end{aligned} \tag{2}$$

Step 3 Event $\{A_s = i \neq i_s^*, N_s(\gamma, i) \geq L\}$ occurs only in one of the following three cases:

a. arm i is substantially overestimated: $\bar{X}_s(\gamma, i) > \mu_s(i) + c_s(\gamma, i)$

b. the best arm $*$ is substantially under-estimated: $\bar{X}_s(\gamma, *) < \mu_s(*) - c_s(\gamma, *)$

c. arm i and the best arm $*$ are too close: $\mu_s(*) - \mu_s(i) < 2c_s(\gamma, i)$

Which means:

$$\begin{aligned}
P(A_s = i \neq i_s^*, N_s(\gamma, i) \geq L) \leq\ & P\left(\bar{X}_s(\gamma, i) > \mu_s(i) + c_s(\gamma, i)\right) \\
&+ P\left(\bar{X}_s(\gamma, *) < \mu_s(*) - c_s(\gamma, *)\right) \\
&+ P\left(\mu_s(*) - \mu_s(i) < 2c_s(\gamma, i), N_s(\gamma, i) \geq L\right)
\end{aligned} \tag{3}$$

For $L = \frac{16\xi \log n_T(\gamma)}{(\Delta\mu_T(i))^2}$, we have

$$\begin{aligned}
\mu_s(*) - \mu_s(i) - 2c_s(\gamma, i) &= \mu_s(*) - \mu_s(i) - 4\sqrt{\frac{\xi \log n_s(\gamma)}{N_s(\gamma, i)}} \\
&\geq \mu_s(*) - \mu_s(i) - 4\sqrt{\frac{\xi \log n_s(\gamma)}{L}} \\
&\geq \mu_s(*) - \mu_s(i) - \Delta\mu_T(i) \geq 0
\end{aligned} \tag{4}$$

so that the third event $\mu_s(*) - \mu_s(i) < 2c_s(\gamma, i)$ never occurs. Due to the regularly reducing assumption, $\frac{X_s(i)}{c_i}$ is a stationary process,

$$\begin{aligned}\bar{X}_t(\gamma, i) &= \frac{1}{N_t(\gamma, i)} \sum_{s=1}^{t} \gamma^{t-s} \varepsilon_i^{s-1} X_s(i) I(A_s = i)/c_i \\ &= \frac{1}{N_t(\gamma, i)} \sum_{s=1}^{t} \frac{\gamma^t X_s(i)}{\varepsilon_i c_i} \left(\frac{\varepsilon_i}{\gamma}\right)^s I(A_s = i) \\ &\leq \frac{1}{N_t(\gamma, i)} \sum_{s=1}^{t} \gamma^{t-s} \frac{X_s(i)}{c_i} I(A_s = i)\end{aligned} \tag{5}$$

According to **Lemma**, by picking $\eta = 4\sqrt{1-\xi/2}$, we upper-bound the probability of the two first events:

$$P\left(\bar{X}_s(\gamma, i) > \mu_s(i) + c_s(\gamma, i)\right) \leq \left\lceil \frac{\log n_s(\gamma)}{\log(1+\eta)} \right\rceil n_s(\gamma)^{-1} \leq \left\lceil \frac{-\log(1-\gamma)}{\log(1+\eta)} \right\rceil \frac{1-\gamma}{1-\gamma^s}$$

Hence, we upper-bound the second sum in the equation(1):

$$\begin{aligned}\sum_{s=K+1}^{T} I(A_s = i \neq i_s^*, N_s(\gamma, i) \geq L) &\leq 2 \sum_{s=K+1}^{T} \left\lceil \frac{-\log(1-\gamma)}{\log(1+\eta)} \right\rceil \frac{1-\gamma}{1-\gamma^s} \\ &\leq 2 \left\lceil \frac{-\log(1-\gamma)}{\log(1+\eta)} \right\rceil \frac{(T-K)(1-\gamma)}{1-\gamma^{K+1}}\end{aligned} \tag{6}$$

Combining equation (1)(3)(6), we obtain the statement of the Theorem. ■
Lemma For stationary rewards gain, the probability :

$$\mathrm{P}(\bar{X}_t(\gamma, i) > \mu_t(i) + c_t(\gamma, i)) \leq \left\lceil \frac{\log n_t(\gamma)}{\log(1+\eta)} \right\rceil n_t(\gamma)^{-2\xi\left(1-\frac{\eta^2}{16}\right)}$$

where $\eta > 0$, $n_t(\gamma) = \sum_{s=1}^{t} \gamma^{t-s} = \frac{1-\gamma^t}{1-\gamma}$ if $\gamma < 1$ [7].

Stopping Time of DUCB-BF Algorithms

Based on the regret analysis above, we can easily prove the following throrem. The following theorem characterizes the lower bound of the expected stopping time of the proposed DUCB-BF algorithms

Theorem 2 For budget B, the stopping time of the DUCB-BF algorithms

$$T(B) \geq \frac{B + (B/c_0 - \widetilde{N}_T(*))(c_0 - c^*)}{c_0}$$

where $* = \underset{i \in \{1,\ldots,K\}}{\arg\max} \frac{\mu_i}{c_i}$, $c_0 = \max_{i \in \{1,\ldots,K\}} c_i$

4 Application and Results

The theoretical regret bound guarantees that DUCB-BF algorithm performs well when the reward is reducing regularly. In this section, an optimization problem on model uncertainty reduction problem is investigated. The object is to minimize the total uncertainty of 'K' models where the model selection policy follows DUCB-BF.

$$\arg \min_{x_k \sim a_k \in A} \sum_{k=1}^{K} |\sigma^2_{k,n_k} - \sigma^2_{k,n_k-1}|,$$

$$s.t. \sum_{k=1}^{K} \sum_{i=1}^{n_k} c_{t_i,k} \leq B$$

- A bandit with $K = 4$ arms is created, which means there are 4 different recovery models (corresponding to 4 maintenance actions).
- The action is to select one of the models to get a new data point x_k, which is used to train and update the model.
- For simplicity, assume model uncertainty m_k is Gaussian distributed. To obtained an unbiased estimation on m_k, we need to get a stable σ^2_k for each 'k' with minimum sample of x_k, where σ^2_{k,n_k} is the variance of maximum likelihood estimation on model m_k.
- The reward is the difference of estimated variance $|\sigma^2_{k,n_k} - \sigma^2_{k,n_k-1}|$. n_k is the training sample number for model 'k'.
- Fixed cost on 4 actions $c_k = [30, 70, 100, 300]$, with total budget B.

The selection strategy for DUCB-BF is:

$$D_{k,t} = \xi_{k,t} \frac{\hat{\mu}_{k,t}}{c_k} + \frac{1}{c_k} \sqrt{\frac{\ln(t-1)}{n_{k,t}}};$$

where $\xi_{k,t} = \frac{|\mu_{k,t} - \mu_{k,t-1}|}{\mu_{k,t-1}}$ is the discount factor. $\hat{\mu}_{k,t}$ is the empirical averaged reward, c_k is the cost of arm 'k', and $n_{k,t}$ is the number of arm 'k' visited at time t.

For algorithms, including UCB [1], UCB-BV (variable cost and using low bound on cost for reward rate) [14], fractional-KUBE (reward rate using individual c_k) [13], and our proposed DUCB-BF, are compared with regret. For any fixed turn T: regret $R_T = T\mu^* - \sum_{t=1}^{T} \mu_{k(t)}$, where $\mu^* = \max_{k=1,\ldots,N} \mu_k$ is the expected reward from the best arm. Before calculate regret for each algorithm, we need to calculate the optimal reward as follow.

- Initial optimal reward $R = 0$ and cost $C = 0$.
- Start with $t = 2$, calculate $\mu_{k,t} = \frac{|\sigma_{k,t} - \sigma_{k,t-1}|}{c_k}$ for $k = 1, \ldots, N$.
- Select arm k^* such that $k^* = \max_k \mu_{k,t}$. Save the reward $R = R + |\sigma_{k^*,t} - \sigma_{k^*,t-1}|$.

- Update cost $C = C + c_{k^*}$.
- Iterate with t until budget runs out, where $\sum_{t=2}^{t^*(B)} c_{k_t^*,t} \leq B \leq \sum_{t=2}^{t^*(B)+1} c_{k_t^*,t}$ (k_t^* is the optimal arm at time t).

The regret for each algorithm is calculated as follows. Let us assume we use algorithm a to select arm.

- Initial reward $R_a = 0$ and cost $C_a = 0$.
- Calculate reward for each arm once, where $\mu_{k,1} = \frac{|\sigma_{k,2}-\sigma_{k,1}|}{c_k}$, where $t = 2N$.
- Select arm based on policy $D_{a,(k,t)}$.
- Based on selected arm a_t (where a_t is the index of the arm pulled by algorithm a at time t), save the reward $R_a = R_a + |\sigma_{a_t,t} - \sigma_{a_t,t-1}|$.
- Update cost $C_a = C_a + c_{a_t,t}$.
- Stop time $t_a(B)$ is to ensure $\sum_{t=2}^{t_a(B)} c_{a_t,t} \leq B \leq \sum_{t=2}^{t_a(B)+1} c_{a_t,t}$ [14].
- The regret for algorithm a is under same budget value B_t, $REG_t = R_{B_t} - R_{a,B_t}$.

We also calculate the regret of group query (query all 4 action at each time) using budget value B_g (accumulated group query cost for each iteration), $REG_g = R_{B_g} - R_{g,B_g}$.

Figure (1) shows the regret on different query strategies. The magenta line is the regret of group query, the green line is the regret of UCB, a blue line is superimposed by green line (only different in low budget) is the regret of UCB-BV, the black line is the regret of fractional-KUBE, and the red line is our proposed algorithm DUCB-BF. From this figure we can see that when arm cost is significantly different, using the reward rate is a better decision, which has less regret when budget is small. Our algorithm beats other algorithm in considering the regert. This suggests that when having discounted reward and finite budget, selecting the arm following the DUCB-BF policy is the best choice. The reason that the benefit is not significant may due to the reward reduces quickly. Also, the discounted rate between previous step and current step might not represent the real situation well.

5 Conclusion and Future Work

Discounted reward on a pre-visited arm is quite common in selection problem. On the other hand, finite budget on query makes the problem and solution more practical in many real application, including gaming and policy generation. To understand the phenomena and study the theoretical meaning of this problem bring this paper. The non-stationary multi-armed bandit problems with budget constraint and fixed costs is explored in this paper, with proposed DUCB-BF algorithm. Theoretical regret bound is studied with a clearly proof lower bound. A real application is investigated with empirically compared regret bound confirmed our finding. This suggests that the proposed DUCB-BF algorithms is a good policy for discounted MAB problem with budget constraint.

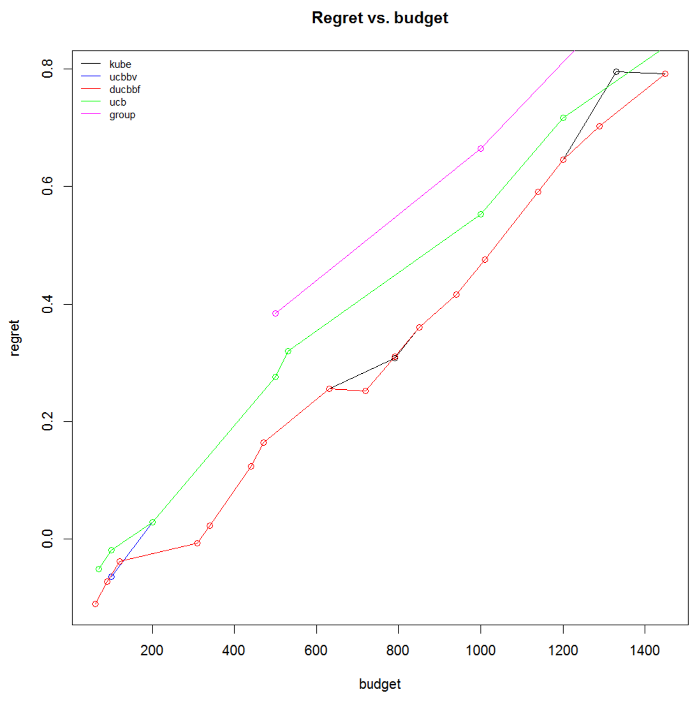


Fig. 1. Regret comparison on Group, UCB, fractional-KUBE, UCB-BV, DUCB-BF query strategies.

Unlike typical discounted MAB problem, our algorithm dose not require extra assumption on reward distribution, which suggests that many real application may apply this algorithm. The regret between different MAB algorithms shrink quickly may due to quickly reduced rewards and variance of the reward converges quickly. This suggests that the DUCB-BF plays well in early selection stage or some problem where the reward variance converges slowly. Also, running this algorithm with more arms (e.g. 100+ models) with highly various costs may need to be explored in next step. Considering the time varying discounted rate $\varepsilon_{i,t}$ wit linear or nonlinear coefficient (or using moving window average) can be another interesting direction.

References

1. Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer, *Finite-time analysis of the multiarmed bandit problem.* Machine learning 47.2-3 (2002): 235-256.
2. Auer, Peter, *Using confidence bounds for exploitation-exploration trade-offs.* The Journal of Machine Learning Research 3 (2003): 397-422.

3. Ansell, P. S. , K. D. Glazebrook, J.E. Nio-Mora, and M. O'Keeffe, *Whittle's index policy for a multi-class queueing system with convex holding costs.* Math. Meth. Operat. Res. 57, 2139, 2003.
4. Ehsan, N., and M. Liu, *On the optimality of an index policy for bandwidth allocation with delayed state observation and differentiated services.* In Proceedings of IEEE INFOCOM, March 2004, Hong Kong.
5. Langford, John and Tong Zhang, *The Epoch-Greedy Algorithm for Contextual Multi-armed Bandits.* NIPS 2007.
6. Auer, Peter, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E. Schapire, *The non-stochastic multiarmed bandit problem.* SIAM journal on computing 32, no. 1 (2002): 48-77.
7. Garivier, Aurélien, and Eric Moulines, *On upper-confidence bound policies for non-stationary bandit problems.* arXiv preprint arXiv:0805.3415 (2008).
8. Babaioff, Moshe, Yogeshwer Sharma, and Aleksandrs Slivkins, *Characterizing truthful multi-armed bandit mechanisms.* In Proceedings of the 10th ACM conference on Electronic commerce, pp. 79-88. ACM, 2009.
9. Radlinski, Filip, Robert Kleinberg, and Thorsten Joachims, *Learning diverse rankings with multi-armed bandits.* In Proceedings of the 25th international conference on Machine learning, pp. 784-791. ACM, 2008.
10. Ny, Jerome Le, Munther Dahleh, and Eric Feron, *Multi-UAV dynamic routing with partial observations using restless bandit allocation indices.* In American Control Conference, 2008, pp. 4220-4225. IEEE, 2008.
11. Li, Lihong, Wei Chu, John Langford, and Robert E. Schapire, *A contextual-bandit approach to personalized news article recommendation.* In Proceedings of the 19th international conference on World wide web, pp. 661-670. ACM, 2010.
12. Dudik, Miroslav, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang, *Efficient Optimal Learning for Contextual Bandits.* UAI 2011.
13. Tran-Thanh, Long, Archie Chapman, Alex Rogers, and Nicholas R. Jennings, *Knapsack based optimal policies for budget-limited multi-armed bandits.* arXiv preprint arXiv:1204.1909 (2012).
14. Ding, Wenkui, Tao Qing, Xu-Dong Zhang, and Tie-Yan Liu, *Multi-Armed Bandit with Budget Constraint and Variable Costs.* AAAI, 2013.
15. S. Bubeck and N. Cesa-Bianchi, *Regret analysis of stochastic and nonstochastic multi-armed bandit problems.* Foundations and Trends in Machine Learning, vol. 5, no. 1, pp. 1122, 2012.

Evolutive deep models for online learning on data streams with no storage

Andrey Besedin[1], Pierre Blanchart[1], Michel Crucianu[2], and Marin Ferecatu[2]

[1] Laboratoire d'Analyse de Données et Intelligence des Systèmes, CEA Saclay, 91191 Gif-sur-Yvette Cedex, France
[2] Centre d'études et de recherche en informatique et communications, Le CNAM, 292 rue Saint-Martin, 75003 Paris, France
andrey.besedin@cea.fr, pierre.blanchart@cea.fr, michel.crucianu@cnam.fr, marin.ferecatu@cnam.fr

Abstract. In recent years Deep Learning based methods gained a growing recognition in many applications and became the state-of-the-art approach in various fields of Machine Learning, such as Object Recognition, Scene Understanding, Natural Language processing and others. Nevertheless, most of the applications of Deep Learning use static datasets which do not change over time. This scenario does not respond well to a number of important recent applications (such as tendency analysis on social networks, video surveillance, sensor monitoring, etc.), especially when talking about online learning on data streams which require real-time adaptation to the content of the data. In this paper, we propose a model that is able to perform online data classification and can adapt to data classes, never seen by the model before, while preserving previously learned information. Our approach does not need to store and reuse previous observations, which is a big advantage for data-streams applications, since the dataset one wants to work with can potentially be of very large size. To make up for the absence of previous data, the proposed model uses a recently developed Generative Adversarial Network to drive a Deep Convolutional Network for the main classification task. More specifically, we propagate generative models instead of the data itself, to be able to regenerate the historical training data that we didn't keep. We test our proposition on the well known MNIST benchmark database, where our method achieves results close to the state of the art convolutional networks trained by using the full dataset. We also study the impact of dataset re-generation with GANs on the learning process.

Keywords: Deep Learning, Data Streams, Adaptive Learning, GAN

1 Introduction

Most of the recent research in Deep Learning community has been focused on the case of static datasets, where the training data is first acquired and then used to train a model [6]. However, nowadays there is a growing demand for real-time processing and analysis of continuously arriving huge amounts of data. One of the main challenges when dealing with online learning on data streams is that the dataset one wants to process is potentially of a very large size, which raises the issue of storage and memory management during training. Another difficulty is that data should be processed in real time while new samples are still continuously arriving. The problem here is that most of machine learning algorithms converge to solution quite slowly and sometimes need several epochs of training over the whole dataset, which is impossible in the online learning setup. Moreover, not storing and reusing historical data usually results in the phenomenon of catastrophic forgetting [7]. An even bigger problem can be caused by concept drifts [11], more specifically by changes in data distribution in time, which may force the algorithm to over-fit on new data and discard useful information, learned from previous observations.

In this paper we mostly consider the problem of storing previous observations and adapting to changes in data content (e.g. new classes become available). To address the storage problem we make use of Generative Adversarial Networks (GANs) [3] which, in the past few years, acquired increased attention in the Machine Learning community due to their ability to learn data representations and generate synthetic samples that are almost indistinguishable from the original data.

Despite their popularity, GANs until now were almost not studied from the point of view of their ability to generalize outside the training set. In this paper we study the notions of generalizability and representativity of generative models and propose quantitative metrics to evaluate them. By generalizability of generative model we mean its capacity to focus on learning concepts and patterns and become a representation of the data distribution rather than reproducing data samples it encounters during training. The term representativity is used to describe the ability of generative models to represent the original dataset it was trained on with all its internal variability. We also investigate the ability of designed online classification system to adapt to the concept drift caused by incremental appearance of new classes of data during learning.

We validate our method on the MNIST database, often used for benchmarking. This choice was mainly guided by the need to have a well-known benchmark for comparison with both offline and online methods using deep neural networks. Our experiments show that recent generative approaches using GANs have excellent ability to generalize and discard the necessity of storing and reusing historical data to perform online training of deep classification models.

To summarize, the contribution of this work is twofold. First, we propose and evaluate a new network architecture that uses GANs to allow to train online classifiers without storing the incoming data while being able to adapt to new classes in the incoming data stream and at the same time to avoid catastrophic forgetting. Second, we justify the usage of GANs in proposed context and make a quantitative evaluation of how the replacement of real data by generated data influences the learning process.

The rest of the paper is organized as follows: In Sec. 2 we motivate our study and present previous results in online classification using deep neural networks and recent approaches for adaptive multi-task learning, in Sec. 3 we give the detailed presentation of our online classification approach and in Sec. 4 we demonstrate the validation results.

2 Related work

There are very few works in the literature that investigate the possibility to train Deep Neural Networks for online classification on data streams.

In [1] the authors address the problem of exploding storage demand in the online learning setup by using generative capacities of Deep Belief Networks (DBNs). In their approach, the authors train DBNs to generate samples similar to the original data, and then use those samples instead of the real data to train a classifier. The drawback of proposed method is the poor generative performance of DBNs on image datasets. It causes a big decrease in classification accuracy, compared to the offline setting with a static training dataset, and results in the performance being far beyond the current state-of-the-art on the MNIST dataset which they used to benchmark their method.

In a more recent work [9] the authors propose a method, Progressive Networks, designed for effective knowledge transfer across multiple sequentially arriving tasks. The evaluation of presented method is showed for the reinforcement learning problems, but the authors state its possible application to a wide range of Machine Learning challenges. In the proposed approach, a single deep neural network is initialized and trained for the first given task. Each time a new task appears, a new neural network with the same architecture as previous ones is added. At the same time, directed connections from all previous models to the current models are initialized and serve as knowledge transfer functions. During the back-propagation, those connections are updated together with the weights of the current model to adjust the influence of previous models on the current one. The pool of historical models stay unchanged and is only used for the forward pass. The big limitation of this method is that the size of parameter space increases rapidly when new tasks are added.

Another approach to deal with changing environment is proposed in [2]. Introduced model, Pathnet, is represented by a huge Neural Network with embedded agents, that are evolving to find the best matching paths through the network for a current task. After the task is learned and the new one is introduced, parameters of the sub-network containing the optimal pathway are "freezed" not to be modified by back-propagation when training for new tasks. PathNet can be seen as an evolutionary version of Progressive Networks, where the model learns its own architecture during training.

Both PathNet and Progressive Networks approaches showed good results for learning on sequences of tasks and can be considered as a good alternative to fine-tuning to accelerate learning. However, they don't provide a way to solve the problem of data storage for streams since every task for these algorithms should be fully described by its complete environment, which is not the case for data streams with only a part of all data classes available at each time point.

Unlike previously described methods, our model is able to learn incrementally on massive multi-class streaming data, is adaptable to changes in data distribution and has no need in excessive historical data storage.

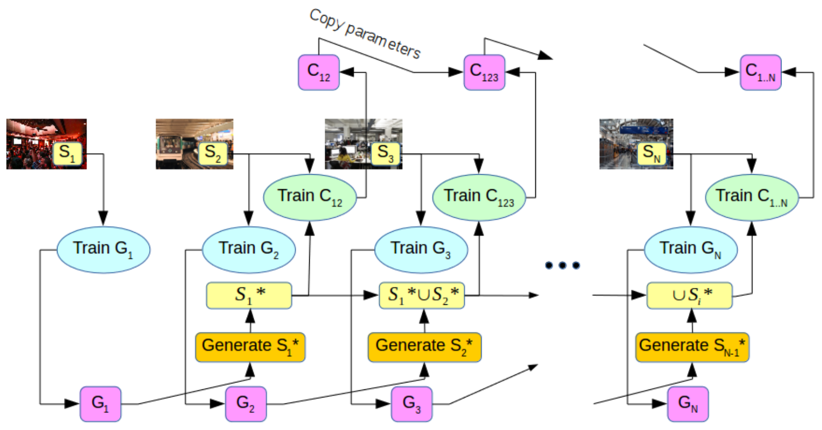


Fig. 1: Schematic representation of our online learning approach. Original data is presented to the model class by class. Each time new class of data appears we start training a new generator modeling that class. At the same time we train a classifier on the generated data from the previously learned classes and the original data from the new class that come from the stream.

3 Proposed approach

To represent the process of online learning we model the streams in a way that the data arrives continuously with distinct classes coming separately one by one. On fig. 1 we show proposed framework.

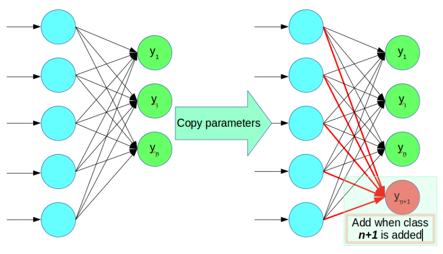


Fig. 2: Adding a node to the output layer and initializing the connections with the previous layer in the online learning scenario when new data class appears in the stream.

Let $S = \{S_i | i = 1, .., N\}$ be the partition of real data into N distinct classes. In our learning framework we take the first coming class S_1 from S and train a generator G_1, able to represent this data. We save G_1 and discard S_1. Then we start training G_2 on the data from S_2, and in parallel train a classifier C_1^2, feeding it with samples from S_1^* – synthetic data, generated by G_1, and newly arriving real data from S_2. After that, data from S_2 are discarded. We continue this procedure with all available classes from S, one by one, each time generating equal batches of data from all the previously trained generators. Each time a new class is added we also add a node to the output layer of the classifier and initialize its connections with the previous layer (fig. 2). The rest of the network weights are copied from the previous state. See algorithm 1 for pseudo-code.

```
Require: $S = \bigcup_{i=1}^{\infty} S_i$ : data stream, with $i$ - class number
Require: $n$ : number of already learned classes
Require: $G_i$ : generative model for class $i$
Require: $C_1^n$ : classification model for data from $\bigcup_{i=1}^{n} S_i$
  $G_1 \Leftarrow$ initialize model
  $n \leftarrow 1$
  while are receiving samples from $S$ do
    $d \Leftarrow$ get batch from $S_j$, $j$ - current class
    if $j = n + 1$ then
      $n \leftarrow n + 1$
      $G_n, C_1^n \Leftarrow$ initialize models
      if $n > 2$ then
        $C_1^n \Leftarrow$ copy parameters from $C_1^{n-1}$
      end if
    end if
    $d^* \leftarrow \bigcup_{i=1..n}^{i \neq j} d_i^*$ generate synthetic data from $\{G_i\}$
    $C_1^n \Leftarrow$ train with $d \bigcup d^*$
    $G_j \Leftarrow$ train with $d$
  end while
```

Algorithm 1: Online learning model, proposed in Sec. 3

Model architecture All the experiments in this paper used a DCGAN [8] model for the generative network, both for the online and offline training settings. DCGANs follow the same training logic as GANs to the difference that DCGAN generator and discriminator networks are built from convolutional layers and have a set of topological constrains to ensure better convergence. Compared to the original GANs, DCGANs show higher stability during training and tend to produce more visually meaningful samples when trained on image datasets.

The classification model hyper-parameters were adjusted according to two criteria:

- the performance of the model should be comparable to the state-of-art
- the network should not be too deep to allow real-time training.

The model we retained consists of one convolutional layer followed by two fully-connected linear layers. Each layer except the last one is followed by a Rectified Linear

Unit activation function. Batch normalization [4] and dropout [10] are applied during training on all layers except the output layer. Stochastic Gradient Descent using Adam [5] is used to perform model parameters optimization. The classification network described in this paragraph is used for all experimental scenarios.

4 Experiments

To test proposed method we used the well-known MNIST[3] dataset of hand-written digits composed of gray-level images of 32x32 pixels, which is classically considered for static off-line approaches. It contains 60000 images in the train set and 10000 images in the test set. The dataset includes 10 classes corresponding to digits from 0 to 9. This database is widely used as a baseline in NN benchmarking and there already exist a lot of pre-trained convolutional models that provide state-of-the-art results on it.

The main goal of this section is to investigate the performance of our approach, more precisely to see how the data generated from DCGANs performs when used to train networks in an online scenario (Sec. 4.3). However before proceeding to that we first make sure that the enabling assumption is correct: that is, we make sure that trained generators are able to represents well the initial training data and generalize on the data distribution (Sec. 4.1 and Sec. 4.2).

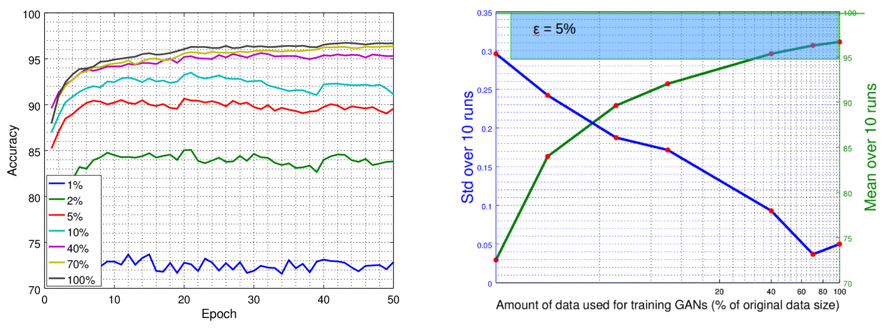
The measure we use in this work to evaluate the fitness of the classifier is the classification accuracy, usually computed as the mean of the normalized diagonal values of the confusion matrix. This measure is well known and used in many research works (for example in [1]).

4.1 Generalizability of GANs

The ability to generalize on the whole data distribution when having only a small number of data samples available for training is one of the main characteristics that any learning system is expected to have. In the case of classification algorithms measuring the generalization capacities of a given model is very straightforward and can be evaluated by the difference between the classification accuracies on the training and validation sets.

Creating a generative model that would focus on learning concepts rather than memorizing single data samples is a problem of a very high importance, since it can be viewed as the machine's capacity to generalize to unseen instances (similar to what creativity and imagination is for human intelligence). In other words, we are more interested in creating a model that would be able to "imagine" objects, that are similar to those from data distribution, rather than just reproducing the data samples it has seen during training, especially in the online learning context where data distributions tend to change in time. This brings us to the question of defining and measuring the generalizability of the generative model. Unfortunately the measure of the model's capacity to generalize cannot be directly transfered from a classification model to a generative model. Instead, we propose a new notion of generalizability that captures much the same principles, but adapted to the generative case.

[3] `http://yann.lecun.com/exdb/mnist/`



(a) (b)

Fig. 3: Results of the generalizability test on MNIST (see Sec. 4.1). (a) Classification accuracy for different GANs support sizes as a function of training time. Average over 10 runs; (b) Mean/std of the classification accuracies for different GANs support sizes over 10 runs after 50 training epochs for the generalizability tests. Blue box represents the area in which the generalization error does not exceed 5%

We shall say that a generative model G trained on some support subset D^{supp} of a dataset D, with D^{val} being the validation set of D such that such that $D^{val} \cap D^{supp} = \emptyset$, generalizes well on D over some measure μ, evaluating the semantical resemblance of samples from two datasets, if the similarity over μ between D^{val} and D^{Gen} - the data sampled from G, approaches the similarity between D^{val} and $D \backslash D^{val}$. In a more formal way:

$$|\mu(D \backslash D^{val}, D^{val}) - \mu(D^{Gen}, D^{val})| < \epsilon,$$

where ϵ is the parameter that determines the quality of generalization.

Choosing the metric to measure the semantical similarity between two datasets is not straightforward. In our study we decided to use a neural network based classification model for this purpose. Since neural networks are known and much appreciated for their ability to learn the abstractions and internal data representations, the mean classification accuracy of this model when trained on one dataset and tested on another one represents a desired property.

One of the main assumptions on the generalization capacities of any machine learning algorithm is that to improve it one often would want to get a bigger training set with more representative data samples. In our experimentations we adopt this idea and adapt it to generative models case.

In following experiment D is the MNIST dataset and G is the set of generative models $G_1, \cdots, G_{10}$ - one for each data class. To evaluate the ability of G to generalize on unseen content we designed a test that consists in varying the size of the set D^{supp} used to train generative models from 60 (1% of D) to 6000 (100%) samples per class and comparing the mean classification accuracy of the classifier, trained on the data generated by G_i, to the one trained and tested on the original data.

Fig. 3a represents classification accuracy as a function of training time averaged over 10 runs of classifier training on generated data G, with G trained on the datasets of different sizes. We can see from the figure that the classification performance on validation set significantly improves when increasing the size of the support set to train the generative models. We observe that with only 1% support size the classification accuracy is pretty high (70-75%) and with a support size of 5% the value goes up to 90%. To quantify this effect, Fig. 3b shows the curve of the improvements through all the tested support size values and makes a link with the generalizability definition proposed earlier, with $\epsilon = 5\%$ and $\mu(D \backslash D^{val}, D^{val}) = 99.6\%$. We can see from the figure that using only 40% of the initial dataset (2400 samples per class) allows us to obtain a highly generalizing generative model with the generalization error below 5% (the values in the blue box). We can also observe that the curve keeps going up which means that having more data for training the generative models should improve the final classification accuracy.

Fig. 4 shows random examples of synthetic samples, generated by DCGAN when trained on different amount of original images from MNIST dataset. We see that starting for 10% support size, the generated data is visually very consistent, confirming the results above.

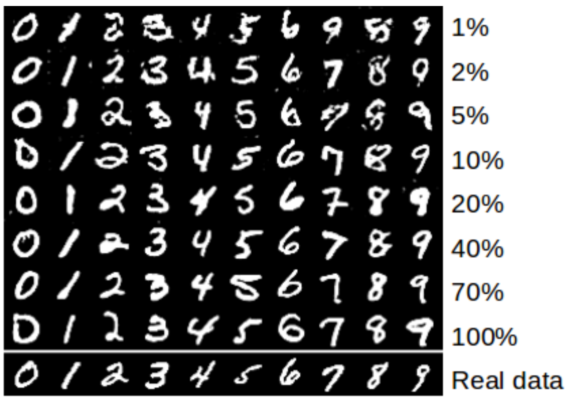


Fig. 4: Samples, produced by DCGAN-based generator, when using 1 to 100% of the original MNIST dataset to train it

4.2 Representativity of Generative Models

Another important question we needed to answer before passing to online scenario is how much data do we need to sample from pretrained generators in order to represent the full richness of the information, learned by generative models from the original dataset. This problem is essential since the amount of data we need to generate while training online classifiers influences directly the learning reactivity and it would be reasonable to sample the smallest amount of data, that at the same time wouldn't affect too much the final classification accuracy.

Similar to the experiment, described in Sec. 4.1 we trained one generative model for each class in MNIST dataset and used the generated data to train a classifier with the difference that this time we used the full dataset as a support for generative models training.

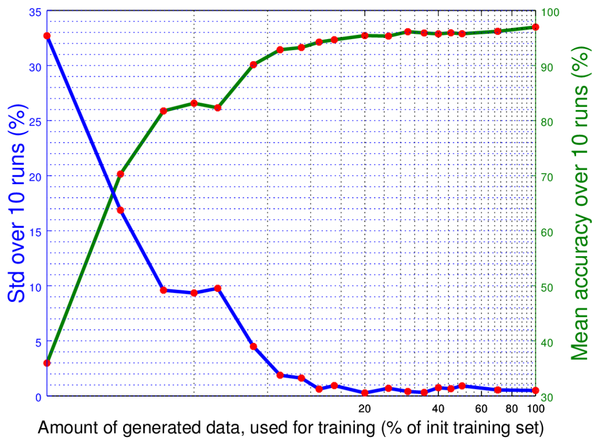


Fig. 5: Results of the representativity test on MNIST (Sec. 4.2). Mean/std of the classification accuracies for different amount of generated data used to train the classifier. Average over 10 runs after 50 training epochs

To verify the representative capacities of DCGAN, we studied the influence of the amount of generated data on the final training accuracy by varying its size from 1 to 100 % of the original dataset. Fig. 5 represents mean and standard deviation of the classification accuracy over 10 runs for each chosen size of generated dataset. We can see that for MNIST dataset, generating samples in the amount of only 30% of the original dataset is enough to reach almost the same training accuracy and stability, represented by low standard deviation, as for the case of generating 100% of the original dataset size.

Comparing to the training on original data, where with the classification model we use we obtain the accuracy of $F^{orig} = 99.6\%$, training on generated data reduces the maximum accuracy down to $F^{gen} = 97.2\%$. We can consider this decrease as the price we pay for not storing the data. Based on these two values we can introduce the metric to evaluate representativity of generative model:

$$R_M(D) = \frac{F_M^{gen}(D^{val})}{F_M^{orig}(D^{val})} = 0.976,$$

where M is the model we evaluate and D^{val} is the validation set - the subset of initial data that was not used to train the generative model. In these notations, the value of $R_M(D)$ close to 1 represents the case of D being well represented by G, we would talk about bad representativity when $R_M(D) << 1$ and $R_M(D) > 1$ corresponds to the case where generative models not only work as the memory to store data representations, but also act as a filtering mechanism that extracts useful information from data samples.

4.3 Online classification using data regeneration

One of the possible limitations of our online classification method, described in Sec. 3, is that to avoid forgetting we need to continuously generate data from all the previously learned classes when receiving samples from new classes. The dependency between the amount of generated data and total number of classes in the dataset may become a problem for classification tasks with large number of classes. On the other hand, synthetic data in our model is used to ensure generalization and stability for learning process and should not be considered as the main source of information for the parameters update. Generating too much of synthetic data can thus reduce the importance that the model gives to original data we receive in streaming mode.

Similar to the tests on representativity of generated models, described in Sec. 4.2, we evaluate the performance of our algorithm depending on the amount of data we generate. The only difference is that in case of data streams we cannot know in advance the size of the dataset, neither the total number of classes.

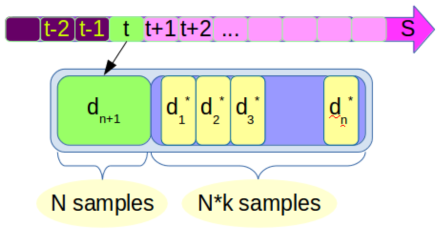


Fig. 6: Schematic representation of the way batches for online training are organized. N is the size of real data batch, coming from stream, n is the number of already learned classes.

To deal with the outlined remarks, we design our experiments in the following way. Each time we receive a batch of stream data of size N, we generate $\frac{\min(n,k)*N}{n}$ data samples for each previously learned class, where k is a parameter, fixed in the beginning of each experiment, and n is the current number of learned classes, so that total volume of generated data is equal to $\min(k,n)*N$ (Fig. 6). The size of generated data batch depends on number of classes we have already learned only when $n \le k$. In each

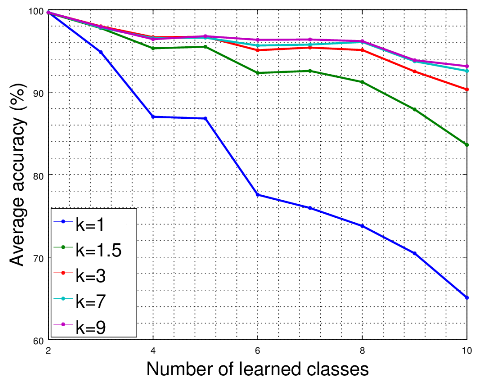


Fig. 7: Accuracy of our online-learning algorithm, described in Sec. 4.3 for different values of scaling parameter k for data regeneration

experiment, fixed value of parameter k in the range between 1 and 9 is taken. Value of 1 corresponds to the case where the total amount of generated data is equal to the size of received batch, while the value of 9 in the 10 classes classification problem represents the case where for each already learned data class we generate the amount of data, equal to the size of received data.

An important aspect to mention is that running the tests directly in streaming mode with just random initialization resulted in a poor performance. To overcome this problem, the classification network was bootstrapped by pre-training for several epochs on the first two data classes and then passing to online mode.

Fig. 7 represents the results of online classification. The graph shows the performance of proposed learning algorithm on the evolving dataset with incrementally added classes of data. Each line is an average over 50 independent runs, corresponding to one of the five tested values of k.

Our method achieves a classification accuracy above 90% in a completely online adaptive scenario starting from $k = 3$, which is close to the state-of-the-art performance in the offline learning setting. The performance increases for higher values of k, i.e., bigger sizes of generated data. As a comparison point, in a similar experimental online setting on the 10 classes MNIST dataset, [1] obtained only 60% accuracy. We can also see that with $k \geq 3$ the accuracy decreases a little with every new added class, but complete forgetting never happens.

5 Conclusion and perspectives

In this work we developed a framework for online training of a Deep Neural Network classifier on data streams with no storage of historical data. The proposed model uses data regeneration with DCGANs to compensate for the absence of the historical data and avoid catastrophic forgetting in the online learning process on data stream.

We justify the choice of DCGAN generative models by showing that they have generalizability and representativity abilities: our experiments confirm that DCGAN-generated data can be used instead of the original data to train a classifier with good generalization properties. These properties allow us to train a classification model that obtains the accuracy above 90% in a completely online learning mode.

In a future part of this work, we will tackle the problem of designing more efficient training methods with less generated data to increase learning reactivity. Training both generative networks and classifier requires indeed having a sufficiently large amount of original data from each presented class, which is often not the case for the real-life applications with dynamic datasets. We will also measure the forgetting effect and find ways to limit its impact. We also plan to validate our online learning scheme on larger and more complex databases.

References

1. R. Calandra, T. Raiko, M. Deisenroth, and F. Pouzols. Learning deep belief networks from non-stationary streams. *Artificial Neural Networks and Machine Learning–ICANN 2012*, pages 379–386, 2012. 3, 6, 11
2. C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017. 3
3. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2
4. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 6
5. D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6
6. Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 1
7. M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165, 1989. 2
8. A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 5
9. A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 3
10. N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 6
11. G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016. 2

Summary Extraction on Data Streams in Embedded Systems

Sebastian Buschjäger, Katharina Morik, and Maik Schmidt

TU Dortmund University
Computer Science VIII: Artificial Intelligence Unit
{sebastian.buschjaeger, katharina.morik}@tu-dortmund.de
http://www-ai.cs.uni-dortmund.de/

Abstract. More and more data is created by humans and cyber-physical systems having sensing, acting and networking capabilities. Together, these systems form the Internet of Things (IoT). The realtime analysis of its data may provide us with valuable insights about the complex inner processes of the IoT. Moreover, these insights offer new opportunities ranging from sensor monitoring to actor control. The volume and velocity of the data at the distributed nodes challenge human as well as machine monitoring of the IoT. Broadcasting all measurements to a central node might exceed the network capacity as well as the resources at the central node or the human attention span. Hence, data should be reduced already at the local nodes such that the submitted information can be used for efficient monitoring.
There are several methods that aim at data summarization ranging from clustering, aggregation to compression. Where most of the approaches transform the representation, we want to select unchanged data items from the data stream, already while they are generated by the cyber-physical system and at the cyber-physical system. The observations are selected independent of their frequencies. They are meant to be efficiently transmitted. The ideal case is that no important measurement is missing in the selection and that no redundant items are transmitted. The data summary is easily interpreted and is available in realtime. We focus on submodular function maximization due to its strong theoretical background. We investigate its use for data summarization and enhance the Sieve-Streaming algorithm for data summarization on data streams such that it delivers smaller sets with high recall.

Keywords: Data summarization, Embedded systems, Streaming Data

1 Introduction

With increasing processing capabilities, more and more data is gathered every day at virtually any place on earth. Most of this data is produced by small embedded electronics with sensing, acting and networking capabilities [11].

To capitalize on the vast amount of data produced by these IoT devices, machine learning has proven a valuable tool, but is usually limited to large server systems due to resource constraints. In order to speed up the analysis for monitoring, we wish to move parts of the analysis to the measuring device by the means of data summarization. For numerical data streams, moving averages have been used. For categorial values, data summarization describes the extraction of representative items from a data set and has been used for texts [7,8], speech [14], and images [13]. Informally, a summary should contain a few, but informative items, where each item reflects one hidden state or class of the underlying system. Given such a summary, we can use it to

– perform certain actions, once a new item is added to the summary
– compare summaries of different devices and points in time
– provide a quick overview for human experts

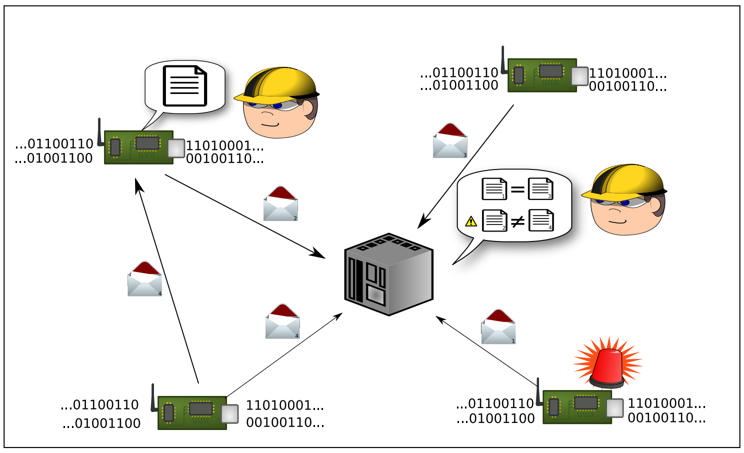


Fig. 1: Use cases of data summarization in IoT streaming settings. Each device gathers a data summary on its own and communicates it to central server or to other IoT devices. Human experts can examine summaries and perform actions if needed.

Data summarization has been viewed as a submodular function maximization problem, which offers an appealing theoretical framework with strong quality bounds. In this paper, we focus on data summarization on data streams by the means of submodular function maximization, namely the Sieve-Streaming algorithm [1].

The main question when computing a data summary is, what items are "novel" enough to be included into the summary and which are too similar to already seen items. If this novelty threshold is chosen too small, the summary will contain redundant items, whereas if the novelty is expected to be too high, we will not add any item to the summary. Sieve-Streaming deals with this challenge by managing multiple summaries in parallel, each with its own novelty threshold.

We will exploit the ideas of Sieve-Streaming for monitoring of distributed measuring devices. Hence, we first investigate, whether the Sieve-Streaming is appropriate for our setting. Additionally, we extend the algorithm for the use in embedded systems:

– We reduce the number of summaries by tighter bounds of the utility threshold used by Sieve-Streaming without sacrificing its theoretical guarantees.
– By dynamic thresholding, we further improve the quality of Sieve-Streaming without increasing its computational costs.
– We evaluate the enhancements empirically.

The paper is organized as the following. The next section will give a brief overview of the Sieve-Streaming algorithm. Section 3 shows our enhancements of the Sieve-Streaming algorithm in detail. Section 4 presents experiments on three different data sets to evaluate data summarization in general and our enhancements of Sieve-Streaming in detail. We conclude the paper in section 5.

2 Sieve-Streaming

In this section, we shortly present the Sieve-Streaming algorithm and introduce the notation used in this paper. A more detailed overview of submodular function maximization can be found in [4]. Submodular function maximization considers the problem of selecting a set $S \subseteq V$ with fixed size $|S| = K$ from a ground set V, such that a set function $f\colon 2^V \to \mathbb{R}$ is maximized, which assigns a utility score to each possible subset $S \subseteq V$. For set functions, the marginal gain is given by the increase in $f(S)$ when adding an element $e \in V$ to S:

$$\Delta_f(e|S) = f(S \cup \{e\}) - f(S)$$

The function f is called monotone, iff for all $e \in V$ and for all $S \subseteq V$ it holds that $\Delta_f(e|S) \geq 0$. Furthermore, we call f submodular iff for all $A \subseteq B \subseteq V$ and $e \in V \setminus B$ it holds that

$$\Delta_f(e|A) \geq \Delta_f(e|B)$$

Submodular functions incorporate a notion of diminishing returns: Adding a new element to a smaller set will increase the utility value more than adding the same element to a larger set. This property intuitively captures one aspect of summarization, in which we seek small but expressive summaries and thus adding elements to a summary should be done with care.

Now, let us apply monotone submodular function maximization in a streaming setting, so that data items e arrive one at a time and we have to decide immediately, whether we want to add e to S, or not. A number of different approaches for streaming settings have been proposed in literature with different constraints and assumptions about the data (see [9] for a very recent overview). The approach of Badanidiyuru and colleagues fits our settings best [1].

Submodularity and monotony allows us to greedily increase the utility value by simply adding new items to the summary. Since the maximum summary size is restricted to K, this approach simply picks the first K elements without considering other items in the stream. Thus, we should add items to the summary only, if they offer a reasonable increase in the summary's utility value, i.e. add enough novelty.

Let $OPT = max_{S \subseteq V, |S|=K} f(S)$ denote the maximum possible function value. Assume we know OPT beforehand and we have already selected some points in S. In this situation we could expect the next item e to add at least $\frac{OPT - f(S)}{K - |S|}$ to the utility so that we can reach OPT with the remaining $K - |S|$ elements. In a sense, this approach sieves out elements with marginal gains below the given threshold.
However, this approach does not work, if the optimal summary contains many elements with a gain just below the threshold and one element with large marginal gain. In this case, we could miss the items with smaller marginal gain waiting for an item with large gain. To counter this behavior, the authors of [1] propose to lower the threshold by $\frac{1}{2}$, adding e to S if the following holds:

$$\Delta_f(e|S) \geq \frac{OPT/2 - f(S)}{K - |S|} \tag{1}$$

Knowing the optimal utility value OPT is part of the problem we would like to solve. Usually, this value is unknown beforehand and needs to be estimated. In order to guarantee a good estimate we can generate multiple estimates for OPT before running the algorithm and manage multiple sieves each with their own threshold values in parallel.
We can narrow down the value of OPT by using the properties of submodularity. To do so, let $m = max_{e \in V} f(\{e\})$ denote the maximum value of a singleton item. Given e occurs at least once in the data stream, the worst optimal summary would contain e at least once. If e however occurs at least K times in the data stream, then the best summary would contain e K-times, so that:

$$m \leq OPT \leq K \cdot m \tag{2}$$

Therefore, knowing the maximum singleton value $f(\{e\})$ already gives a rough estimate of the range of values we can expect from f. This knowledge can now be used to sample different threshold values from the interval $[m, Km]$ such that one of these thresholds will be close to OPT. More formally, we manage different summaries in parallel, each using one threshold from the set $O = \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1+\varepsilon)^i \leq k \cdot m\}$, so that for at least one $v \in O$ it holds: $(1-\varepsilon)OPT \leq v \leq OPT$. Algorithm 1 summaries the proposed method. It can be shown, that algorithm 1 extracts a summary with:

$$f(S) \geq (\frac{1}{2} - \varepsilon)OPT \tag{3}$$

2.1 Utility function

Given a summary $S = \{e_1, \ldots, e_K\}$ we can express the similarity between elements e_i and e_j using a kernel function $k(e_i, e_j)$, which gives rise to the kernel matrix $\Sigma = [k(e_i, e_j)]_{ij}$ containing all similarity pairs. This matrix has the largest

Algorithm 1 Sieve-Streaming with known singleton maximum value m.

```
 1: $O = \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1+\varepsilon)^i \leq k \cdot m\}$
 2: for $v \in O$ do
 3:     $S_v = \emptyset$
 4: end for
 5: for next item $e$ do
 6:     for $v \in O$ do
 7:         if $\Delta_f(e|S_v) \geq \frac{v/2 - f(s)}{K - |S|}$ and $|S_v| < K$ then
 8:             $S_v = S_v \cup \{e\}$
 9:         end if
10:     end for
11: end for
```

values on the diagonal as items are the most similar to themselves, whereas values on the off-diagonal indicate the similarity between distinct elements and thus are usually smaller. Intuitively, we seek an expressive summary, so that Σ contains many values near 0 on its off-diagonal. This intuition is formally captured by the Informative Vector Machine [5] proposing the function:

$$f(S) = \frac{1}{2} logdet(I + \sigma \Sigma)$$

where I indicates the $K \times K$ identity matrix and $\sigma > 0$ denotes some scaling parameter. In [10] it was shown, that this function is monotone submodular.

3 Embedded Summary Extraction

In the previous section, we presented the Sieve-Streaming algorithm. Now, we will introduce the changes that make it better suited for summarization in embedded systems. First we show how to reduce the number of sieves without sacrificing performance guarantees. Second, we introduce dynamic thresholding which further increases the utility value. Last, we consider the trade-off between memory consumption and runtime for an efficient implementation.

Reduce the number of sieves: Sieve-Streaming needs to keep track of multiple sieves in parallel with corresponding thresholds chosen from the interval $[m, Km]$. By taking the special structure of f into account, we can reduce the size of this interval and thus the overall memory requirements. To do so, we assume that the kernel function $k(\cdot, \cdot)$ and its corresponding kernel matrix are positive definite.
The upper bound Km of the threshold v is sharp and thus cannot be improved. Intuitively, a summary reaches its maximum function value when all entries on the off-diagonal are 0. In this case $logdet(I + \sigma^2 \Sigma)$ equals $log((1+\sigma)^K) = Klog(1+\sigma) = K \cdot m$, which is exactly what we derived using submodularity.
The lower bound m, however, can be improved substantially: A summary reaches

its minimum utility when all entries on the off-diagonal equal the largest possible kernel value C, that is, we picked the same element K times. W.l.o.g we assume that $C = 1$ in the following, e.g. we use the RBF kernel $k(x, y) = exp(-\frac{1}{2}||x - y||^2)$. Note, that it is possible to scale every positive definite kernel to the interval $[0, 1]$ without damaging its positive definite property (cf. [3]). By using Sylvester's determinant theorem, we see that $logdet(I + \sigma\Sigma) = log(I+\sigma\mathbf{1}^T\mathbf{1}) = log(1+\sigma K) > m = log(1+\sigma)$ where $\mathbf{1}$ denotes the vector, where all entries are 1. In order to show that this intuition really creates a lower bound, we use the characteristic polynomial $det(\lambda I - (I + \sigma\Sigma)) = \sum_{i=1}^{K}(-1)^k\lambda^{K-i}a_i$. The determinant can be computed in terms of characteristic coefficients a_i, that is $det(I + \sigma\Sigma) = \sum_{i=0}^{K} a_i = 1 + det(\sigma\Sigma) + tr(\sigma\Sigma) + \sum_{i=2}^{K-1} a_i$. Since $I + \sigma\Sigma$ is positive definite, each coefficient a_i can be expressed as the sum of eigenvalues of $I + \sigma\Sigma$, which are also all positive [2]. Since $\sigma\Sigma$ is also positive definite, we see that $det(\sigma\Sigma) \geq 0$. This leads then to

$$det(I + \sigma\Sigma) = 1 + det(\sigma\Sigma) + tr(\sigma\Sigma) + \sum_{i=2}^{K-1} a_i \geq 1 + tr(\sigma\Sigma) = 1 + \sigma K$$

and thus: $logdet(I + \sigma\Sigma) \geq log(1 + \sigma K)$.

Dynamic Thresholding: Sieve-Streaming creates a fixed number of sieves $|\mathcal{O}|$ in the beginning of the algorithm. Due to the thresholding approach, sieves with smaller thresholds will accept more items and thus fill-up more quickly, whereas sieves with large thresholds are more picky. Once a summary is full the corresponding sieve is not used anymore. This allows us to create another sieve in its place with a different threshold while keeping the overall number of sieves constant.

In turn, we can use this new threshold to refine the estimate of OPT. Let v_c denote the largest threshold corresponding to a full summary and let v_o denote the smallest threshold corresponding to a non-full summary. Note that by construction $v_o > v_c$ and that - given the current summaries - OPT seems to be in the interval $[v_c, v_o]$. Therefore, we can refine the estimate of OPT if we create a new sieve in $[v_c, v_o]$.

With this approach “older” sieves with larger threshold may contain more items than “newer” sieves. In turn, one of these “older” sieves may become full at one point, whereas the newly created sieves with smaller threshold are still accepting elements. Since the thresholds for these sieves are smaller than that the sieve that just became full, we know that these sieves will not offer any better utility values. Therefore, we can close these sieves once a sieve with larger threshold closes.

In practice one rather unintuitive effect may happen, in which all summaries of sieves might be full at some point in time. In pure Sieve-Streaming, sieves are created in the interval $[m, Km]$, where the upper bound Km represents the largest utility possible. This bound is independent from the data and therefore we expect that at least one sieve will be open all the time aiming for an extreme case with utility Km.

Recall however, that only the half of each threshold (see eq. 1) is used when the marginal gain of a new item is evaluated. Thus, Sieve-Streaming effectively uses $\frac{1}{2}Km$ as upper bound. This is enough to ensure a $(\frac{1}{2} - \varepsilon)$ approximation, but might be sub-optimal in some cases. To counter this, we gradually increase the interval of thresholds to $2Km$, once we notice that all sieves are closed.

Implementation: In a naive approach, each sieve needs to store K elements and compute $logdet(I + \sigma\Sigma)$ on-the-fly as required. Computation of the determinant is generally performed in cubic time, whereas the computation of Σ takes $\mathcal{O}(K^2 \cdot d)$ leading to a total of $\mathcal{O}(K^3 + K^2 \cdot d)$ per element.
To improve the overall speed, we can trade some runtime with memory if we store Σ permanently. Since $k(\cdot, \cdot)$ is positive definite, $I + \sigma\Sigma$ is also positive definite. Thus, we may use a Cholesky decomposition $I + \sigma\Sigma = L^T L$ where L denotes a lower triangular matrix to compute the determinant.
Adding a new row / column vector to a Cholesky decomposition can be performed in $\mathcal{O}(K^2)$ time. Thus, when adding a new element to the summary, we compute all necessary kernel values and update the Cholesky decomposition accordingly leading to $\mathcal{O}(K^2 + K \cdot d)$ computations per element. However, with this method the memory requirements increase, since Σ and L need to be stored continuously, leading to $\mathcal{O}(2K^2 + K)$ memory instead of $\mathcal{O}(K)$ memory.

4 Experiments

In this section we experimentally evaluate the enhancements introduced to Sieve-Streaming. More specifically, we want to answer the following questions:

1. Are extracted summaries meaningful in a sense, that they represent the hidden states of a system?
2. How does the runtime decrease when a reduced number of sieves is used?
3. How does the utility value increase when dynamic thresholding is used and how does this affect the quality of the summary?

Answering the first question is difficult, because we need to know the data generation process in detail to detect the hidden states of a system. For real-world data this is nearly impossible. Hence, we use data sets $\mathcal{D} = \{(\boldsymbol{x_i}, y_i) | i = 1, \dots, N\}$ usually found in classification tasks. Here, each observation $\boldsymbol{x}_i \in \mathbb{R}^d$ is also associated with a label y_i. We assume that each label y_i represents one hidden state of the system in the data generation process. Thus, we compute a summary of the training data $\boldsymbol{x}_i$ without considering the labels y_i and then report the number of different labels represented by the best summary found across all sieves. With respect to the notation used in classification tasks, we denote this measure as recall.

To answer the second and third question, we measure the computation time per data item and the best utility value of each algorithm (pure, reduced, and dynamic).

All experiments were performed on an Intel i7-6700 machine with 3.4 Ghz and 32 GB RAM. We repeated each experiment 10 times with shuffled data sets and always report average results. Our implementation is available at `https://bitbucket.org/sbuschjaeger/iotstreaming2017`.
Figure 3 contains all experimental results. Each column represents one data set whereas each row represents one measure. We use one synthetic data set and two real-world data sets.

Synthetic data: We composed a synthetic data set containing eight four-dimensional Gaussian distributions as a Gaussian mixture model. Each Gaussian mean value $\boldsymbol{\mu} = (\mu_i)_i$ is uniform randomly generated with $-2 \leq \mu_i \leq 3$ for $i = 1, 2, 3, 4$. Variance values are chosen uniformly random from $(0, 0.8]$.

For the data generation process we first randomly pick one of the Gaussian distributions and then sample a new data item $\boldsymbol{x} \in \mathbb{R}^4$ from this distribution. We repeat this process 10000 times. To simulate infrequent states, e.g. failure, we introduced different probabilities for each Gaussian. Two Gaussian are rare to be selected with a probability of 0.001, whereas the remaining six Gaussian receive the remaining probability mass equally, that is each has a probability of 0.133 to be used for data generation.

We use the RBF Kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2}{10})$ and set $\sigma = 1$ and $\varepsilon = 0.1$. Since there are eight hidden states in total, a summary of size $K = 8$ should be enough in theory to detect all states of the system. To account for some error however, we vary K from 10 to 24.

The first image in the first row of Figure 3 shows the recall of all three algorithms on the synthetic data set. Sieve-Streaming and its tuned counterpart achieve roughly 60% recall for $K = 10$ and then steadily increase their recall with rising K reaching nearly 100% at $K = 20$. Dynamic Sieve-Streaming shows a similar performance, but behaves slightly better for smaller K. Interestingly, for $K = 16$ all variants produce the same output and for $K = 18$ Dynamic Sieve-Streaming is outperformed by pure Sieve-Streaming. For larger K however, Dynamic-Sieve Streaming detects all states with 100% recall for $K = 24$.
Note that a potential "failure" state has a probability of 0.001, that is roughly every 1000 measurements such "failure" may occur. Usually, a human operator would need to examine all 1000 states in this case to determine if there is a failure or not. Using a summary with $K = 24$ however, a human operator only needs to examine 24 different measurments, reducing the overall workload for the human operator by 99.976%.

Looking at the second row in Figure 3, we see that Sieve-Streaming and its implementation on a smaller interval offers the same utility, which can be expected. Dynamic Sieve-Streaming consistently achieves higher utility values explaining the higher recall.

The last row of plots depict the runtime results. Sieve-Streaming needs around 0.45 ms per element, whereas its reduced variant is faster for all K. Dynamic Sieve-Streaming on the other hand needs up to 0.52 ms per element, due to its dynamic sieve management.

UJIndoor Location: As a second data set, we use the `UJIndoorLoc` data set [12]. This data set contains the GPS position and RSSI fingerprints of nearby WiFi hotspots as well as the corresponding building, floor and room number of 17 smartphone users throughout their daily routine at the university of Jaume in Spain. The dataset in total contains 19937 measurements and is usually used for prediction tasks in which one tries to infer the GPS position based on the RSSI fingerprints.

We abandon the RSSI fingerprints and try to predict the semantic location, that is the building, floor and room number directly based on the current GPS position. We normalize the GPS data and use the RBF Kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2}{0.005})$. We set $\sigma = 1$ and $\varepsilon = 0.1$. On average, each user visits 79 different locations, thus we try different K from 80 to 130.
The second column in Figure 3 shows the results for this data set. The recall is displayed in row one. Standard Sieve-Streaming and its reduced counterpart achieve between 50% and 60% recall, whereas Dynamic Sieve-Streaming manages to detect roughly 10% more items offering a recall of 60% to 70%. Again, this increase in recall can be explained with a higher utility value: Dynamic Sieve-Streaming increases the utility value consistently by roughly 10 points compared to Sieve-Streaming. The runtime of the three algorithms (second column and third row of Figure 3) paints a different picture than before. Again, pure Sieve-Streaming is the slowest algorithm, whereas Sieve-Streaming on the smaller interval reduced the overall computational costs by around 0.2ms per element. For smaller K, Dynamic Sieve-Streaming can be found in the middle of the two algorithms. For larger K however, Dynamic Sieve-Streaming seems to become faster than the reduced version of Sieve-Streaming. The reasons for that is, that Sieve-Streaming may exit a data set early if all sieves are full. Since Dynamic Sieve-Streaming reopens new sieves, it will run through the complete data set. This in turn, enables cache locality and branch prediction to take full effect, so that the runtime per element decreases.

MNIST: As a third data set, we use the well-known `MNIST` data set [6]. This data set contains 60000 28×28 grayscale images of the handwritten digits 0 to 9 and is usually used as a baseline measure for image classification. Much in line with the usual classification task, we try to extract a summary containing one representative image for each digit. However, we will do this in an unsupervised manner. Again, we normalize the data and use the RBF kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2}{784})$. We set $\sigma = 1$ and $\varepsilon = 0.1$. Since there are 10 classes in total, we vary K from 8 to 16.
The results are shown in the third column of figure 3. All algorithms performed nearly the same, with Dynamic Sieve-Streaming being slightly better for all values of K reaching a recall between 60 and 80%. Again, the utility value for Dynamic Sieve-Streaming is consistently larger than for pure and reduced Sieve-Streaming explaining the increased recall performance. Figure 2 displays two summaries for $K = 8$ computed by pure Sieve-Streaming and dynamic Sieve-Streaming respectively. Both algorithms show a similar summary, but pure Sieve-

Streaming selects multiple ones and fives. Dynamic Sieve-Streaming also selects multiple ones, but does not choose to include two fives into the summary and thus can add six, seven and eight to the summary increasing the recall significantly. Looking at the runtime, standard Sieve-Streaming is again the slowest algorithm variant reaching a computation time of 1ms per element, whereas reduced and dynamic Sieve-Streaming are both faster with a maximum of 0.65 ms per element. Note, that we again observe the effect, that Dynamic Sieve-Streaming seems to be faster than its reduced counterpart due to cache locality and branch prediction.

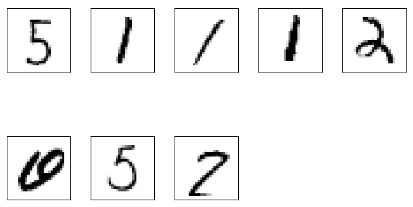
5 Conclusion

In this paper we tackled the problem of data summarization on data streams with small, embedded systems. More specifically, we tried to answer the following questions:

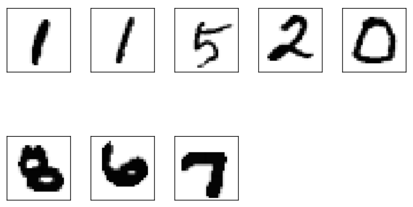
– Is submodular function maximization a suitable framework for data summarization and can summarization help us to detect the current state of a system?
– How can we make data summarization on data streams more suitable for small embedded systems frequently found in IoT settings?

In order to answer these questions, we first presented the Sieve-Streaming algorithm from [1]. Sieve-Streaming works on the assumption of submodularity and uses this to sieve out unwanted items. The main advantage of Sieve-Streaming is its strong and universal theoretical foundation offering the approximation guarantee of at least $(\frac{1}{2} - \varepsilon)$.
This theoretical foundation however, leaves room for a more specialized version of Sieve-Streaming tailored towards data summarization. Therefore, we introduced a more refined version of Sieve-Streaming for data summarization by reducing the overall number of Sieves needed. Additionally, we introduced a dynamic version of Sieve-Streaming independent from the summarization task which is able to reuse sieves with different thresholds. We note, that both enhancements do not jeopardize the theoretical analysis of Sieve-Streaming and still maintain the $(\frac{1}{2} - \varepsilon)$ approximation guarantee.
We evaluated our algorithmic changes on three data sets. The experiments have shown that we are able to detect the states of systems using the presented data summarization approach and that an increased utility function usually comes with an increased recall of system states. Thus, Dynamic Sieve-Streaming is a valuable enhancement of the standard Sieve-Streaming algorithm consistently increasing the recall leading to a 99.976% reduction of workload for human operators in our experiments.

Dynamic Sieve-Streaming has its limitations. We did not always detect all hidden states in the experiments. Future work will investigate other kernel functions, which could have been tuned more.



(a) Pure Sieve-Streaming



(b) Dynamic Sieve-Streaming

Fig. 2: Extracted summary for MNIST using pure Sieve-Streaming and Dynamic Sieve-Streaming for $K = 8$.

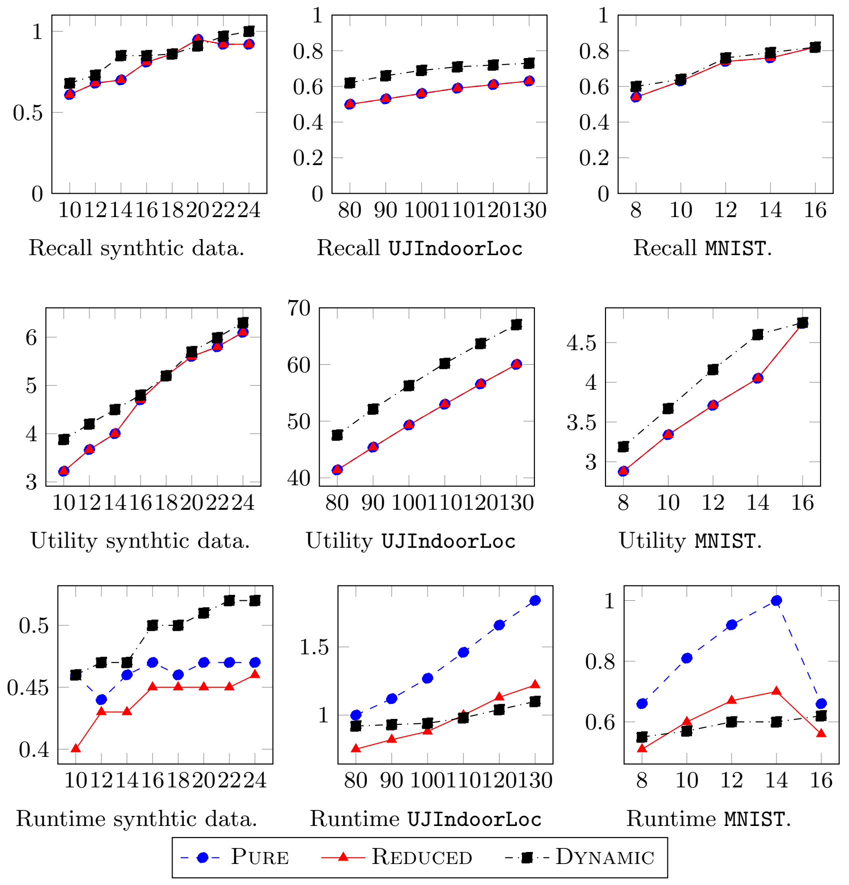


Fig. 3: Experiments on synthetic data (left column), UJIndoor location data (middle column) and MNIST (right column). The first row depicts recall (higher is better), the second row shows the maximum utility value (higher is better) and the third row displays the runtime (in milliseconds) per element (smaller is better) on the datasets.

Additionally, we observe that Sieve-Streaming and its newer enhancements do not deal with concept drift. Here, further work is also needed.

In sum, dynamic Sieve-Streaming is a helpful tool for exploring and monitoring data streams with a solid theoretical foundation. At the same time, we also see a potential for new research directions including concept drift and specialized kernel functions.

Notes and Comments. Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 (`http://sfb876.tu-dortmund.de`) "Providing Information by Resource-Constrained Analysis", project A1.

References

1. Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., Krause, A.: Streaming submodular maximization: Massive data summarization on the fly. In: ACM SIGKDD (2014)
2. Brooks, B.P.: The coefficients of the characteristic polynomial in terms of the eigenvalues and the elements of an n× n matrix. Applied mathematics letters (2006)
3. Graf, A.B., Borer, S.: Normalization in support vector machines. In: DAGM Symposium of Pattern Recognition (2001)
4. Krause, A., Golovin, D.: Submodular function maximization. In: Tractability: Practical Approaches to Hard Problems. Cambridge University Press (2014)
5. Lawrence, N., Seeger, M., Herbrich, R., et al.: Fast sparse gaussian process methods: The informative vector machine. In: NIPS (2003)
6. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
7. Lin, H., Bilmes, J.: A class of submodular functions for document summarization. In: Meeting of the Association for Computational Linguistics (2011)
8. Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A.: Fast constrained submodular maximization: Personalized data summarization. In: ICML (2016)
9. Mirzasoleiman, B., Karbasi, A., Krause, A.: Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In: ICML (2017)
10. Seeger, M.: Greedy forward selection in the informative vector machine. Tech. rep., University of California at Berkeley (2004)
11. Stolpe, M.: The internet of things: Opportunities and challenges for distributed data analysis. ACM SIGKDD Explorations Newsletter (2016)
12. Torres-Sospedra, J., Montoliu, R., Martínez-Usó, A., Avariento, J.P., Arnau, T.J., Benedito-Bordonau, M., Huerta, J.: Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In: IPIN (2014)
13. Tschiatschek, S., Iyer, R.K., Wei, H., Bilmes, J.A.: Learning mixtures of submodular functions for image collection summarization. In: NIPS (2014)
14. Wei, K., Liu, Y., Kirchhoff, K., Bilmes, J.A.: Using document summarization techniques for speech data subset selection. In: HLT-NAACL (2013)

Hybrid Self Adaptive Learning Scheme for Simple and Multiple Drift-like Fault Diagnosis in Wind Turbine Pitch Sensors

Houari Toubakh and **Moamar Sayed-Mouchaweh**
IMT Lille Douai, Univ. Lille, Unite de Recherche Informatique Automatique, F-59000 Lille, France
e-mail: houari.toubakh@imt-lille-douai.fr

Abstract

This paper presents a hybrid dynamic data-driven approach to achieve simple and multiple drift like fault detection of pitch system sensors. This approach considers the system evolving in non-stationary environments and switching between several control modes. This switching is entailed by changes in the system environments. In each control mode, the system has a different dynamical behavior. The latter is described in a feature space sensitive to normal operating conditions in the corresponding control mode. These operating conditions are represented by restricted zones in the feature space called classes. The latter are characterized by a set of parameters representing their statistical properties, e.g. gravity center and variance-covariance matrix. The occurrence of an incipient fault entails a drift in the system operating conditions until the failure takes over completely. This drift manifests as a progressive change in the classes parameters in each control mode over time. The proposed approach monitors normal classes parameters in order to detect a drift in their characteristics. This drift detection allows achieving the fault in its early stages. It uses two drift indicators. The first indicator detects the drift and the second indicator confirms it. Both indicators are based on the observation of changes in the normal operating conditions characteristics over time. A wind turbine simulator is used to validate the performance of the proposed approach.

1 INTRODUCTION

The search for alternative clean energy is undoubtedly becoming more and more important in modern societies. The growing interest in wind energy production has led to the design of sophisticated wind turbines (WTs). Like every other complex and heterogeneous system, WTs are faced to the occurrence of faults that can impact their performance as well as their security. Therefore, it is crucial to design a reliable automated diagnostic system in order to achieve fault detection and isolation in early stage.

Fault diagnosis of WTs is a challenging task because of the high variability of the wind speed and the confusion between faults and noises as well as outliers. However, the fault diagnosis of pitch system is particularly a challenging task because of (i) the occurrence of pitch system faults in power optimization zone in which the fault consequences are hidden and (ii) the actions of the control feedback which compensate the fault effects. The role of the pitch system is to adjust the pitch of a blade by rotating it depending on the pitch angle position reference provided by the controller. The latter decides the pitch angle position reference according to the wind speed in order to allow an optimum energy production.

In the literature, there are several methods [6],[9],[11],[12],[1],[4],[15] that are used to achieve fault diagnosis in WTs. They achieve the fault diagnosis by reasoning over differences between desired or expected behavior, defined by a model, and observed behavior provided by sensors. They can be classified into two main categories of methods: internal and external methods. The internal methods [17],[18],[20] use a mathematical or structural model to represent the relationships between measurable variables by exploiting the physical knowledge or/and experimental data about the system dynamics. These variables represent the internal parts of the wind turbine. The response of the mathematical model is compared to the observed values of variables in order to generate indicators used as a basis for the fault diagnosis. Generally, the model is used to estimate the system state, its output or its parameters. The difference between the system and the model responses is monitored. Then, the trend analysis of this difference can be used to detect changing characteristics of the system resulting from a fault occurrence. The internal methods used to achieve the fault diagnosis of wind turbines are divided into three main categories: parameter estimation [8],[19], observer and state estimation based [3],[23] and signal analysis or feature based [7],[21] approaches. These methods were applied successfully to achieve the diagnosis of faults impacting the pitch system [19],[3],[16], the generator [16],[14], the converter [25],[14], and the gearbox [26],[16].

The major advantages of these methods are their ability to detect both the abrupt and progressive failures via trend analysis, and they give a precise decision or isolation of a failure. However, they suffer from the necessity to depth information about system behavior and failures which is hard to obtain for complex and strong non-stationary systems as wind turbines.

An alternative to overcome this problem is the external methods [17],[22],[9]. The external methods consider the system as a black box, in other words, they do not need any mathematical model to describe the system dynamical behaviours. They use exclusively a set of measurements or/and heuristic knowledge about system dynamics to build a mapping from the measurement space into a decision space. They include expert systems and machine learning and data mining techniques. These methods are suitable for systems that are difficult to model, they are simple to implement and require short processing time. However, since the obtained models are not transparent, the obtained results are hard to be interpreted and demonstrated. There are several machine learning and data mining methods used to achieve the fault diagnosis of wind turbines. Such methods are described and successfully applied in [24],[2].

Few approaches have been proposed to achieve early fault diagnosis of WTs, in particular pitch sensors. This is due to the fact that modeling component degradation in strong non-linear and complex non-stationary environments is very hard task. Examples of these methods, we can cite genetic algorithm [10], neural network, the boosting tree algorithm, and support vector machine [9]. These methods do not integrate a mechanism to detect a drift by analyzing the characteristics of incoming data and to update the model parameters and structure in response to this drift. Therefore, they do not achieve a reliable early diagnosis. Consequently, the diagnosis performance (diagnosis delay) is decreased significantly for faults occurring in WT critical subsystems as pitch systems ones.

This paper presents a new data-driven based approach in order to achieve a reliable drift monitoring and diagnosis of simple and multiple drift-like faults that can affect wind turbine pitch sensors. This approach takes into account the different dynamical behaviors of WTs according to the wind speed. The goal is to detect a drift from normal operating conditions using only the recent and useful data. Initial offline modeling allows constructing initial classes based on the historical data set. These classes characterize the operating conditions of the pitch system (normal/faulty) and are represented by restricted zones in the feature space. The latter is formed by sensitive features to pitch sensor operating conditions in order to distinguish any drift from normal to fault operating conditions. The modeling tool is an algorithm called AuDyC (Auto-Adaptive Dynamical Clustering) used to initialize the classes that will be dynamically updated.

In this work, two-dimensional feature space is constructed, for the sensor faults. The faulty classes, representing the failure operating conditions of pitch sensor, are considered to be a priori unknown. There is one known class in advance.The class represents the pitch sensor normal operating conditions. It considers gradual degradations in pitch sensor operating condition as a drift in the characteristics of normal class over time. Detecting and following this drift can help to predict the occurrence of pitch sensor failure.

The drift-like fault is monitored using two drift indicators: one to detect a drift and the second one to confirm it. When the drift is detected by the first indicator, a warning is emitted to human operators. Then, the second drift indicator confirms this drift in order to inform human operators of the necessity to react by taking the adequate correction actions.

The proposed data-driven approach is composed of five main steps: processing and data analysis, clustering and classification, drift monitoring, updating and interpretation steps.

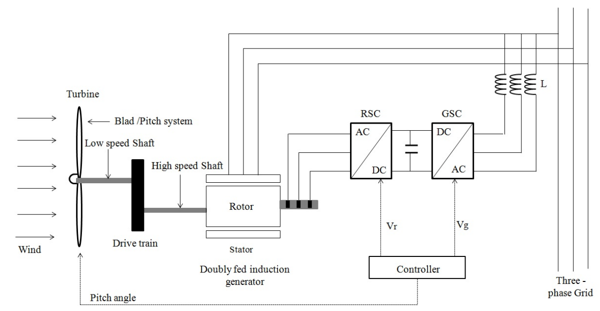


Figure 1: Wind turbine components.

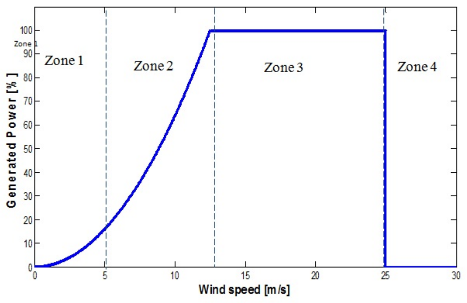


Figure 2: Reference power curve for the WT depending on the wind speed.

2 Pitch system within wind turbines

The wind turbine model under study is composed of five principal parts: the blades, the drive train, the generator with the converter, and the controller (see Figure 1). It can be seen that the blades are fixed to the main axis, which in turn is connected to the generator through the drive train. The generator is electrically connected to the converter, which in turn is connected to a transformer. The blades are pitched by the pitch actuators.

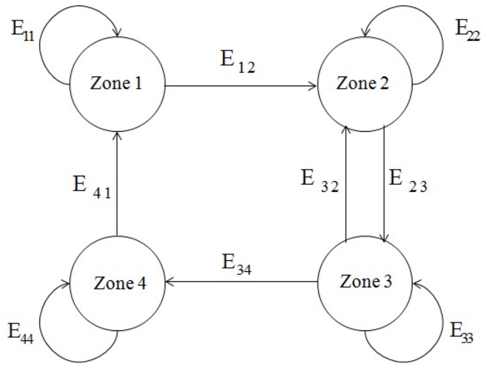

$$\begin{tikzcd}
\text{Zone 1} \arrow[loop left, "E_{11}"] \arrow[r, "E_{12}"] & \text{Zone 2} \arrow[loop right, "E_{22}"] \arrow[d, shift left, "E_{23}"] \\
\text{Zone 4} \arrow[loop left, "E_{44}"] \arrow[u, "E_{41}"] & \text{Zone 3} \arrow[loop right, "E_{33}"] \arrow[l, "E_{34}"] \arrow[u, shift left, "E_{32}"]
\end{tikzcd}$$

Figure 3: Controller operating zones modeled by a finite state automaton.

The controller operates in four zones (see Figure 2). Zone 1 is the start-up of the turbines, zone 2 is power optimization, zone 3 is constant power production and zone 4 is no power production due to a too high wind speed.

In order to handle transitions between the control modes, the controller checks the operating zone in which the WT is by observing the wind speed. The transitions between the control modes change the dynamics of the pitch system. Each control mode is active in one zone thus it is modeled by a finite state automaton. Each zone is represented by a state in which a specific control mode or strategy is defined. According to the wind speed, the control mode changes by switching from one mode or state to another mode or state. This switching between control modes is achieved by discrete events. As an example, if the WT was initially in control mode related to the zone 1, as long as the wind speed is less than a predefined threshold (5 m/s in Figure 2) E_{11} will be generated. E_{11} keeps the WT in control mode 1. If the wind speed is greater than the predefined threshold for zone 1 (5 m/s in Figure 2), The event E_{12} is generated leading to switch the WT from the control mode related to zone 1 to the control mode related to zone 2 (see Figure 3). Same reasoning can be applied for the other events.

The focus of this benchmark model is on the operation of WT in zones 2 and 3. Two control strategies are applied to optimize the energy production and keep it constant at its optimal value: the converter torque control in zone 2 and the blades angle control in zone 3 (see Figure 4). In zone 2, the WT is controlled so that it produces as much energy as possible. To do so, the blades angle is maintained equal to 0° and the tip speed ratio is kept constant at its optimal value. The latter is regulated by the rotating speed control by tuning the converter torque. Once the optimal power production is achieved, the blades angle control maintains the converter torque constant and adjusts the rotating speed by controlling the blades angle. The latter modifies the transfer of the aerodynamic power of the wind on the blades. In this work, the controller modes are modeled by a finite state automaton containing two states (see Figure 4). In the following, zones 2 and 3, respectively, correspond to control modes 1 and 2:

Control Mode 1 In this control mode, the power optimum value is achieved by setting the pitch reference to zero $\beta[t] = 0$ and the reference torque to the converter $\tau_{g,r}$ as follows:

$$\tau_{g,r} = K_{opt} \times \left(\frac{\omega_g[t]}{N_g}\right)^2 \tag{1}$$

N_g is the gear ratio and n is the sampling time.
Where

$$K_{opt} = \frac{1}{2}\rho A R^3 \frac{C_{P_{\max}}}{\lambda^3_{opt}} \tag{2}$$

with ρ the air density, A the area swept by the turbine blades, $C_{P_{max}}$ the maximum value of power coefficient, and λ_{opt} the optimal value of λ is found as the optimum point in the power coefficient C_P mapping of the WT. The power coefficient mapping characterizes the efficiency of energy and it depend on λ and β.

Control Mode 2 In this mode, the major control actions are handled by the pitch system using a Proportional Integral (PI) controller trying to keep $\omega_g[t]$ at ω_g.

$$\beta_r(t) = \beta_r(t-1) + k_p.e(t) + (k_i.T_s.k_p).e(t-1) \tag{3}$$

When $e(t) = \omega_r(t) - \omega_{nom}$. In this case the converter reference is used to suppress fast disturbances:

$$\tau_{g,r}(t) = \frac{P_r(t)}{\omega_t(t)} \tag{4}$$

The control mode should switch from mode 1 to mode 2 if the following condition is satisfied:

$$E_{23} : \omega_g(t) \geq \omega_{nom} \tag{5}$$

The satisfaction of this condition generates a discrete event, E_{23}, allowing the switching from control mode 1 to control mode 2. The goal to obtain P_g equal to P_r. This condition is satisfied when the wind speed is greater than predefined threshold for zone 2 (12.5 m/s in Figure 2). Likewise, the control mode should switch from control mode 2 to control mode 1 if the following condition is satisfied:

$$E_{32} : \omega_g(t) < \omega_{nom} - \omega_\Delta \tag{6}$$

Where ω_{nom} is the nominal generator speed and ω_Δ is a small offset subtracted from the nominal generator speed to introduce some hysteresis in the switching scheme, thereby avoiding that the control modes are switching all the time [15]. The satisfaction of this condition generates a discrete event, E_{32}, allowing the switching from control mode 2 to control mode 1. This condition is satisfied when the wind speed is less than the wind speed threshold defined for zone 3 (12.5 m/s in Figure 2).

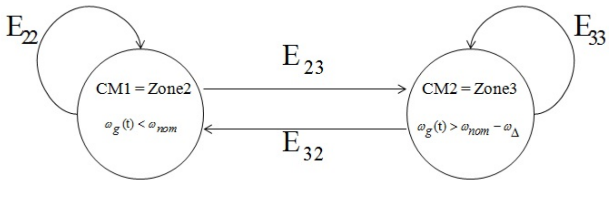


Figure 4: Controller modes modeled by a finite state automaton

As we said before, the benchmark model allows simulating the WT behavior in two power zones: 1) zone 2 (power optimization) where τ_g is controlled and β_r is equal to zero and; 2) zone 3 (optimal energy production) where τ_g is kept β_r constant and is controlled. In this paper, we focus on pitch sensor faults as it is discussed in subsection 2.

3 Pitch system description

The considered WT is horizontal-axis based with three blades. Each blade is equipped with an actuator. The role of the pitch actuator is to adjust the pitch of a blade by rotating it; Each actuator is provided by the same pitch angle reference β_r. The pitch angle of a blade is measured on the cylinder of the pitch actuator, each pitch position (angle) β_{m_i} where $i \in \{1, 2, 3\}$ is measured with two sensors where index m_i represents the i^{th} sensor of the corresponding variable (see Figure 5). The pitch system feedback β_f is an internal variable used to model the pitch position error caused by sensor faults:

$$\beta_f = \beta_r - \frac{1}{2}(\beta_{k,m1} + \beta_{k,m2}) \tag{7}$$

The controller is fed by the mean value of the readings of the two sensors. Hence, this sensor fault is modeled as a change in the pitch references, meaning that a sensor fault resulting in changed mean value should also change the pitch reference accordingly [15].

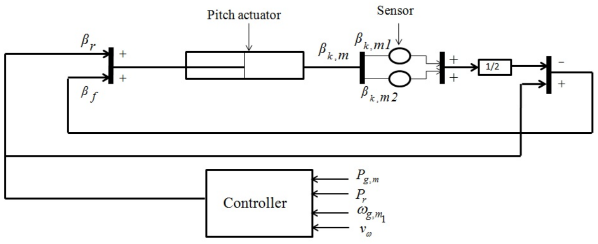


Figure 5: Block diagram of pitch system for the blade k, $(k = 1, 2, 3)$

4 Pitch system modeling

The hydraulic pitch system is modeled in the benchmark as a closed loop of dynamic system. The state representation of the nominal pitch system dynamics is defined as follows [15]:

$$\begin{aligned} \dot{x}_p &= A_p x_p + B_p(\beta_r + \beta_f) \\ y_p &= C_p x_p \\ A_p &= \begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\zeta\omega_n \end{bmatrix} \\ B_p &= \begin{bmatrix} 0 \\ \omega_n^2 \end{bmatrix} \\ C_p &= \begin{bmatrix} 0 & 1 \end{bmatrix} \end{aligned} \tag{8}$$

The state vector $x_p = \begin{bmatrix} \dot{\beta}_k & \beta_k \end{bmatrix}^T$ is composed of pitch angular speed $\dot{\beta}_k$, and position β_i for each blade $k : (k = 1, 2, 3)$. y_p is the measured pitch position, β_r is the pitch angle position reference provided by the controller, and β_r is the feedback pitch system (see Figure 5). ω_n, ζ are the parameters of the pitch system where ω_n represent the natural frequencies and ζ is the damping ratio.

The pitch system represent a hybrid dynamic system and especially it belongs to the class of Discretely Controlled Jumping Systems (DCJS), In these systems, the continuous state variables change discontinuously under the influence of an external action (e.g., a command) as the case for electromagnetic systems with pulse inputs [?]. The pitch system state variable $x_p = \begin{bmatrix} \dot{\beta}_k & \beta_k \end{bmatrix}^T$ changes discontinuously under the influence of an external action defined by Equation 5 and 6.

5 Pitch system drift-like fault scenarios generation

In this paper the types of fault which are considered in this work are simple and multiple drift-like fault in pitch sensors. The following subsections detail the generation of several scenarios representing drift-like faults with three different speeds in pitch sensor β_{m1} and pitch sensor β_{m2}, and in both pitch sensors β_{m1} and β_{m2}.

5.1 Sensor drift-like fault

Each blade is equipped with an actuator. Each actuator is provided by the same pitch angle reference β_r. In addition, each pitch position, (angle) β_{mi} is measured with two sensors where index i represents the i^{th} sensor of the corresponding variable. The fault scenarios related to simple drift-like fault in pitch sensor $n°1$ and sensor $n°2$ and multiple drift-like fault in both pitch position sensor $n°1$ and sensor $n°2$ in blade $n°3$ are summarized respectively in Table 1, Table 2 and Table 3. The state representation of the pitch system after the integration of a fault in sensor β_{mi}, $i \in \{1, 2\}$ is defined as follow:

$$\begin{aligned} \dot{x_p} &= Ax_p + Bu \\ y_p &= Cx_p + f(t) \\ f(t) &= \lambda_i.(t_b - t_e) \end{aligned} \tag{9}$$

Therefore the parameter λ_i, $i \in \{1, 2\}$ is used in the simulation to generate a fault in sensor β_{mi} during the time period $(t_b - t_e)$ where t_b is the start time and t_e is the end time of sensor drift-like fault.

Simple drift-like fault in sensor β_{m1}

In this paper the simple drift-like fault scenarios in pitch sensor 1 (β_{m1}) scenarios are modeled as a gradual change in the coefficient λ_1 of pitch sensor $n°1$ in blade $n°3$ where t_b is the beginning of the drift and t_e is the end of the drift. Nine scenarios for simple sensor drift-like fault are generated in order to simulate slow, moderate and high degradation speeds represented by slow, moderate and high drift speeds (see Figure 6). Each drift speed scenario is generated at three different time instances. Thus, parameter λ_1 is changed linearly from λ_{1N} to λ_{1F} in a period of 30s, 60s and 90s, corresponding respectively to high, moderate and slow drift speeds. Then, the fault remains active for 200s. Finally the parameter λ_1 decreases again to return to its initial value λ_{1N} (see Figure 6 for the case of high drift speed in sensor 1 (β_{m1})).

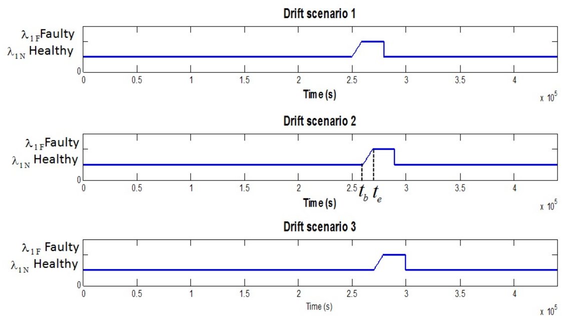


Figure 6: Simple drift-like fault scenarios in pitch sensor 1 (β_{m1}), corresponding to high drift speed in 3 different time instances t_b is the beginning time of the drift and t_e is the end of the drift.

Simple drift-like fault in sensor β_{m2}

The simple drift-like fault scenarios in pitch sensor 2 (β_{m2}) scenarios are modeled as a gradual change in the coefficient

Fault N°	Drift speed	Simple drift-like fault in pitch sensor β_{m1}	Period
F_{4h}	30s (High)	$\lambda_{1N} \rightarrow \lambda_{1F}$	2500s -2730s
F_{4m}	60s (Medium)	$\lambda_{1N} \rightarrow \lambda_{1F}$	2500s -2760s
F_{4s}	90s (Slow)	$\lambda_{1N} \rightarrow \lambda_{1F}$	2500s -2790s
F_{5h}	30s	$\lambda_{1N} \rightarrow \lambda_{1F}$	2600s -2830s
F_{5m}	60s	$\lambda_{1N} \rightarrow \lambda_{1F}$	2600s -2830s
F_{5s}	90s	$\lambda_{1N} \rightarrow \lambda_{1F}$	2600s -2890s
F_{6h}	30s	$\lambda_{1N} \rightarrow \lambda_{1F}$	2700s -2930s
F_{6m}	60s	$\lambda_{1N} \rightarrow \lambda_{1F}$	2700s -2960s
F_{6s}	90s	$\lambda_{1N} \rightarrow \lambda_{1F}$	2700s -2990s

Table 1: Simple drift-like fault scenarios in pitch sensor 1 (β_{m1}).

λ_2 of pitch sensor $n°2$ in blade $n°3$ where t_b is the beginning of the drift and t_e is the end of the drift. As for the case of simple drift-like fault in pitch sensor β_{m1} scenarios, nine scenarios for simple sensor drift-like fault are generated in order to simulate slow, moderate and high degradation speeds represented by slow, moderate and high drift speeds (see Figure 7). Each drift speed scenario is generated at three different time instances. Thus, parameter λ_2 is changed linearly from λ_{2N} to λ_{2F} in a period of 30s, 60s and 90s, corresponding respectively to high, moderate and slow drift speeds. Then, the fault remains active for 200s. Finally the parameter λ_2 decreases again to return to its initial value λ_{2N} (see Figure 7 for the case of high drift speed in sensor 2, (β_{m2})).

Fault N°	Drift speed	Simple drift-like fault in pitch sensor β_{m2}	Period
F_{7h}	30s (High)	$\lambda_{2N} \rightarrow \lambda_{2F}$	2800s -3030s
F_{7m}	60s (Medium)	$\lambda_{2N} \rightarrow \lambda_{2F}$	2800s 3060s
F_{7s}	90s (Slow)	$\lambda_{2N} \rightarrow \lambda_{2F}$	2800s- -3090s
F_{8h}	30s	$\lambda_{2N} \rightarrow \lambda_{2F}$	2900s -3130s
F_{8m}	60s	$\lambda_{2N} \rightarrow \lambda_{2F}$	2900s -3130s
F_{8s}	90s	$\lambda_{2N} \rightarrow \lambda_{2F}$	2900s -3190s
F_{9h}	30s 30s	$\lambda_{2N} \rightarrow \lambda_{2F}$	3000s -3230s
F_{9m}	60s	$\lambda_{2N} \rightarrow \lambda_{2F}$	3000s -3260s
F_{9s}	90s	$\lambda_{2N} \rightarrow \lambda_{2F}$	3000s -3290s

Table 2: Simple drift-like fault scenarios in pitch sensor 2 (β_{m2}).

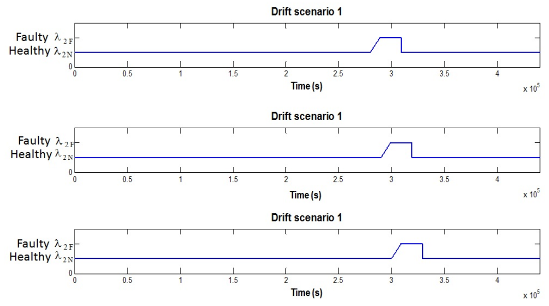


Figure 7: Simple drift-like fault scenarios in pitch sensor 2 (β_{m2}), corresponding to high drift speed in 3 different time instances.

Multiple sensor drift-like fault

In this chapter the generated scenarios of the multiple drift-like fault in pitch sensor 1 (β_{m1}) and sensor 2 (β_{m2}) are modeled as a gradual change at the same time in the drift coefficient (λ_1 and λ_2) of both pitch sensors $n°1$ and pitch sensors $n°2$ in blade $n°3$. As for the case of simple drift-like fault in pitch sensor scenarios, nine scenarios for multiple sensor drift-like fault are generated in order to simulate slow, moderate and high degradation speeds representing by slow, moderate and high drift speeds (see Table 3). Each drift speed scenario is generated at three different time instances. Thus, parameters λ_1 and λ_2 are changed linearly from λ_{1N} and λ_{2N} to λ_{1F} and λ_{2F} in a period of 30s, 60s and 90s, corresponding respectively to high, moderate and slow drift speeds. Then, the fault remains active for 200s. Finally the parameter decreases again to return to their initial values (see Figure 8 for the case of high drift (degradation) speed in both sensor 1 (β_{m1}) and sensor 2 (β_{m2})).

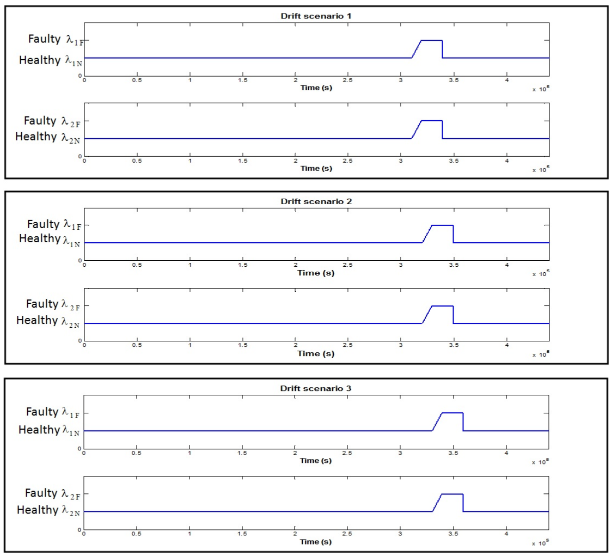


Figure 8: Multiple sensor drift-like fault scenarios in sensors (β_{m1}) and (β_{m2}) corresponding to high drift speed in 3 different time instances.

Fault N°	Drift speed	Multiple drift-like fault in in pitch sensors β_{m2} and β_{m2}	Period
F_{10h}	30s (High)	$\lambda_1 N \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3100s-3330s
F_{10m}	60s (Medium)	$\lambda_1 N \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3100s-3360s
F_{10s}	90s (Slow)	$\lambda_1 N \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3100s-3390s
F_{11h}	30s	$\lambda_{1N} \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3200s-3430s
F_{11m}	60s	$\lambda_{1N} \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3200s-3460s
F_{11s}	90s	$\lambda_{1N} \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3200s-3490s
F_{12h}	30s	$\lambda_{1N} \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3300s-3530s
F_{12m}	60s	$\lambda_{1N} \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3300s-3560s
F_{12s}	90s	$\lambda_{1N} \rightarrow \lambda_{1F}$ and $\lambda_{2N} \rightarrow \lambda_{2F}$	3300s-3590s

Table 3: Multiple drift-like fault scenarios in pitch sensors (β_{m1}) and (β_{m2}).

6 Proposed approach

In this section, hybrid dynamic data-driven approach is developed in order to achieve condition monitoring and drift like fault detection of pitch sensor. It performs predictive diagnosis by detecting a drift of the system operating conditions from normal to faulty modes. The proposed approach is based on 5 steps developed in the following subsections (see Figure 9).

6.1 Processing and data analysis

This step aims at finding the features that are sensitive to the system operating conditions in order to construct the feature space. A feature space representing the operating conditions of each assembly of WT is defined, this feature space will be responsible of the detection and isolation of faults impacting this components. The research of sensitive features is based on the signals provided by the pitch sensors as well as the prior knowledge about the system dynamics. These features are chosen in order to maximize the discrimination between operating conditions in the feature space. In this paper, two-dimension feature space is constructed for the sensor fault. The goal of the feature space use, at the level of component, is to facilitate the drift-like fault isolation and to enhance the diagnosis robustness.

The position of the pitch actuators is measured by two redundant sensors for each of the three pitch positions $\beta_{k,mi}$, $k = 1, 2, 3$, $i = 1, 2$, with the same reference angle β_r provided to each of them. In order to enhance the robustness against noise, the measurements are filtered by a first order filter using time constant $\tau = 0.06$.

For the drift like fault detection and isolation of the sensor faults, we propose to explore the physical redundancy in order to generate residuals as follows:

$$\Delta\beta_{s1} = |\beta_r + \beta_f - \beta_{m1}| \quad (10)$$

$$\Delta\beta_{s2} = |\beta_r + \beta_f - \beta_{m2}| \quad (11)$$

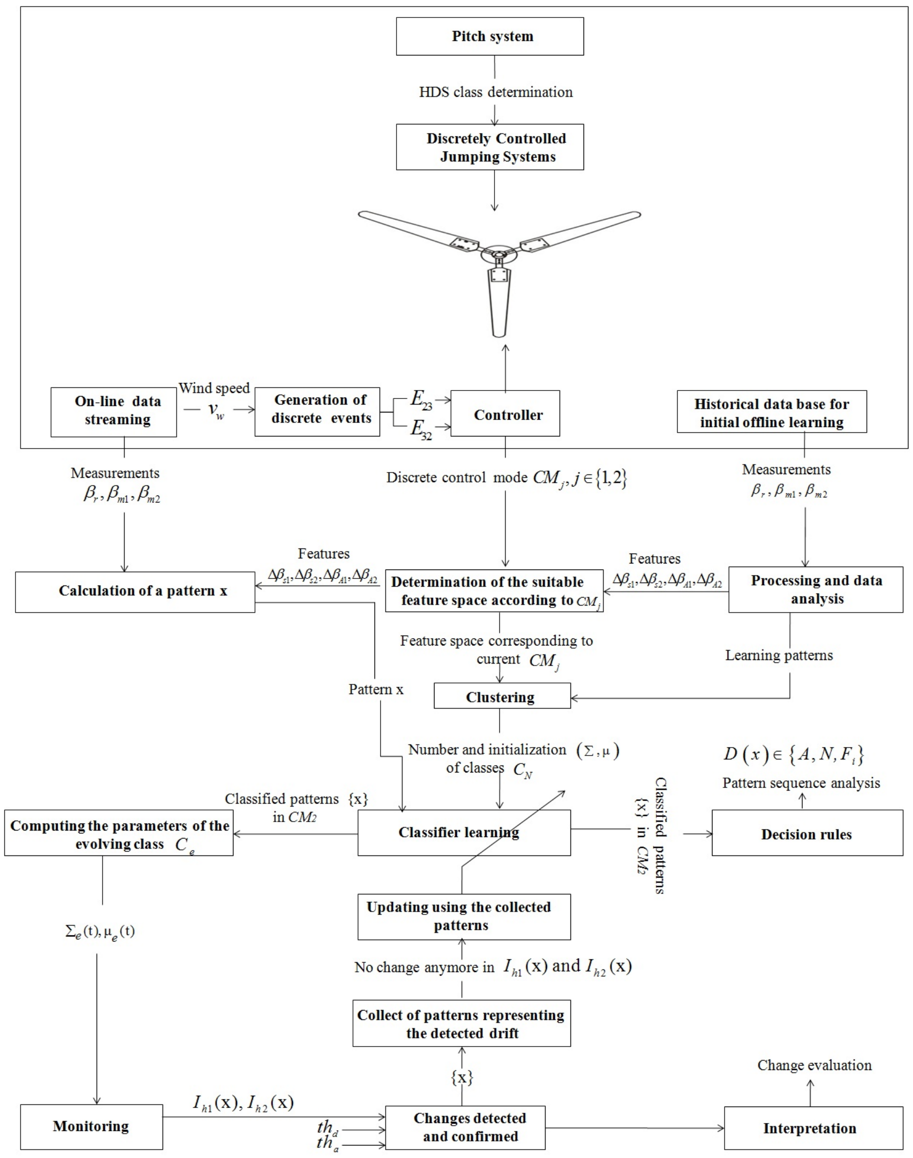
To do so, the residual $\Delta\beta_{sn}$, $n = 1, 2$, is generated by the comparison between the pitch angle measurement β_{mi}, $i = 1, 2$, $m = 1, 2, 3$ and the command computed by the sum of the desired value of the pitch angle β_r and the feedback pitch system β_f (see Figure 5). The residual is computed within a time window which is tuned to be several times the actuator time response.

The evolution of these residuals with respect to each of the two sensors is considered as meaningful features. Indeed, the residual $\Delta\beta_{s1}$ respectively $\Delta\beta_{s2}$, is equal to zero when the corresponding sensor β_{m1} respectively β_{m2}, is in normal operating conditions. When, the sensor β_{m1} respectively β_{m2}, is in faulty operating conditions, the residual $\Delta\beta_{s1}$, $\Delta\beta_{s2}$ will be different of zero because this sensor will not measure the new value of command $(\beta_r + \beta_f)$ (see Figure 5). Indeed, the command $(\beta_r + \beta_f)$ will change in order to compensate the difference between the two sensors due to the fault of sensor β_{m1} respectively β_{m2}.

6.2 Classifier learning and updating

The clustering looks to determine the number of classes contained in the learning set and to initialize their parameters. The classification aims at designing a classifier able to assign a new pattern to one of the learnt classes in the feature space. A new pattern characterizes the actual operating conditions (normal or faulty in response to the occurrence of a certain fault) of the system. Examples of these approaches are present in [5] as well as in the references of this paper.

Auto-adaptive Dynamical Clustering Algorithm (AuDyC) [13] is selected in this work in order to achieve both clustering and classification. AuDyC computes the parameters of initial classes based on the statistical properties of data which are the mean and the variance-covariance matrix. These classes characterize the normal operating conditions of pitch sensors. AuDyC was chosen because it is unsupervised classification method and is able to model streams of patterns since it always reflects the final distribution of patterns in the features space. It uses a technique that is inspired from the Gaussian mixture model [13]. Let E^d be a d-dimensional feature space. Each feature vector $x \in E^d$ is called a pattern. The patterns are used to model Gaussian prototypes P^j characterized by a center $\mu_{P^j} \in R^{d\times 1}$ and a covariance matrix $\sum_{P^j} \in R^{d\times d}$. Each Gaussian pro-



Pitch system
HDS class determination
Discretely Controlled Jumping Systems
On-line data streaming
Wind speed v_w
Generation of discrete events
E_{23}
E_{32}
Controller
Historical data base for initial offline learning
Measurements $\beta_r, \beta_{m1}, \beta_{m2}$
Discrete control mode $CM_j, j \in \{1,2\}$
Measurements $\beta_r, \beta_{m1}, \beta_{m2}$
Features $\Delta\beta_{s1}, \Delta\beta_{s2}, \Delta\beta_{A1}, \Delta\beta_{A2}$
Calculation of a pattern x
Determination of the suitable feature space according to CM_j
Features $\Delta\beta_{s1}, \Delta\beta_{s2}, \Delta\beta_{A1}, \Delta\beta_{A2}$
Processing and data analysis
Feature space corresponding to current CM_j
Learning patterns
Pattern x
Clustering
Number and initialization of classes C_N (Σ, μ)
$D(x) \in \{A, N, F_i\}$
Pattern sequence analysis
Classified patterns {x} in CM2
Computing the parameters of the evolving class C_e
Classifier learning
Classified patterns {x} in CM2
Decision rules
Updating using the collected patterns
$\Sigma_e(t), \mu_e(t)$
No change anymore in $I_{h1}(x)$ and $I_{h2}(x)$
Collect of patterns representing the detected drift
{x}
Change evaluation
Monitoring
$I_{h1}(x), I_{h2}(x)$
th_d
th_a
Changes detected and confirmed
Interpretation

Figure 9: Proposed on-line adaptive scheme steps.

totype characterizes a class. A minimum number of N_{win} patterns are necessary to define one prototype, where N_{win} is a user-defined threshold. A class models operating conditions and gathers patterns that are similar one to each other. The similarity criterion that is used is the Gaussian membership degree. Faults will affect directly this distribution and this will be seen through the continuously updated parameters. More details about AuDyC related to merging classes, splitting classes, rules of recursive adaptation, similarity criteria, etc., can be found in [13].

In the sensor feature space, four classes are considered: the fault of sensor 1, β_{m1} , the fault of sensor 2, β_{m2}, the fault of both sensor 1,β_{m1} and sensor 2 β_{m2}, and the normal functioning. Figure 10 shows the classes representing normal and failure operating conditions of pitch sensor in the feature space constituted by the two residuals defined by Equation 10 and 11. In zone 2, the effects of this fault are hidden because the actuators are not operated. Moreover, it is strongly difficult to distinguish the fault occurrence to the noise in the case of small angles. Therefore an overlapping region is created between the normal and failure classes (see Figure 10 and Figure 15).

In order to answer the challenges inherent to the system operation, the normal and failure classes are split into five classes and the pitch actuator dynamics are represented by two different control modes. The first one corresponds to the case of zone 2 low wind speed; while the second control mode represents the case of zone 3 high wind speed (see Figure 16). Class 1 is the ambiguity class. It gathers the patterns representing pitch sensor normal or faulty operating conditions. This class represents the control mode 1. Class 2 represents the normal operating conditions class in

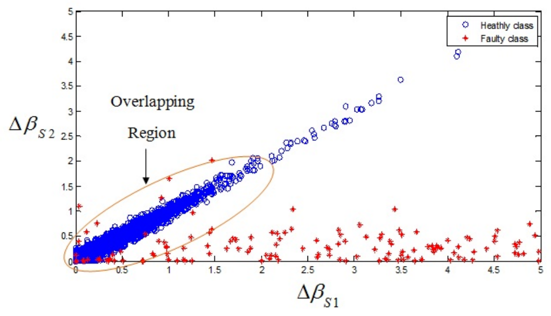


Figure 10: Large view of overlapping region for the pitch sensor normal and failure operating conditions in case of simple fault in pitch sensor 1, (β_{m1}).

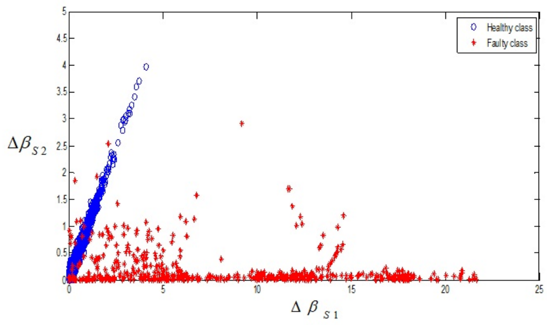


Figure 11: Feature space of the pitch sensor normal and failure operating conditions in case of simple fault in pitch sensor 1, (β_{m1}).

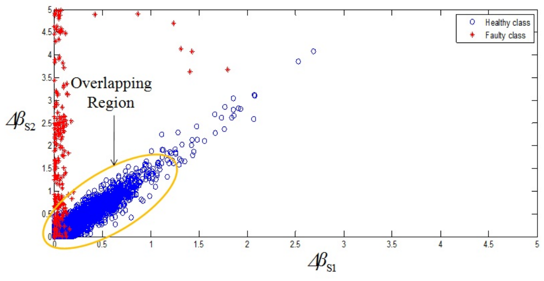


Figure 12: Large view of overlapping region for the pitch sensor normal and failure operating conditions in case of simple fault in pitch sensor 2, (β_{m2}).

control mode 2. Class 3 represents failure class caused by simple drift-like fault in pitch sensor 1, β_{m1} in control mode 2, class 4 represents failure class caused by simple drift-like fault in pitch sensor 2, β_{m2} in control mode 2 and class 5 represents failure class caused by multiple drift-like fault in pitch sensor 1, β_{m1} and sensor 2, β_{m2} in control mode 2.

The updating step aims at reacting to the changes in classes characteristics in the feature space. AuDyC continuously updates the classes parameters by using the recursive adaptation Rules 12 and 13. In such a way, its validity and

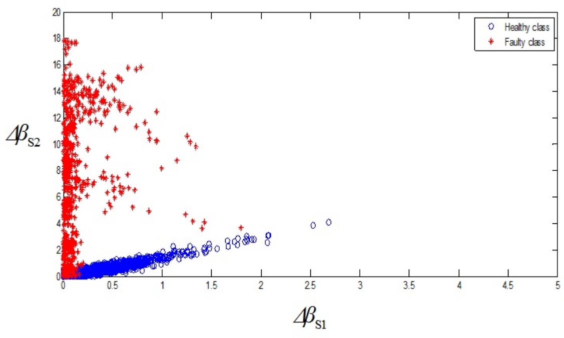


Figure 13: Feature space of the pitch sensor normal and failure operating conditions in case of simple fault in pitch sensor 2, (β_{m2}).

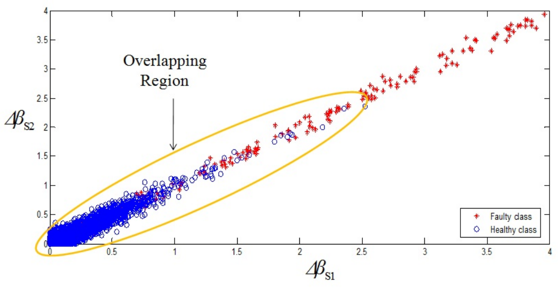


Figure 14: Large view of overlapping region for the pitch sensor normal and failure operating conditions in case of multiple fault in pitch sensor 1, (β_{m1}) and pitch sensor 2, β_{m2}.

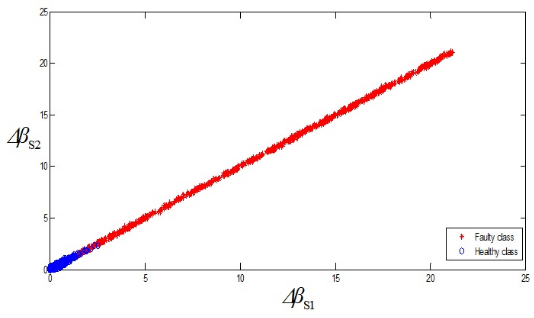


Figure 15: Feature space of the pitch sensor normal and failure operating conditions in case of multiple fault in sensor β_{m1} and β_{m2}.

performance over time is preserved.

$$\mu_e(t) = \mu_e(t-1) + f(\mu_e(t-1), x^{new}, x^{old}, N_{win}) \quad (12)$$

$$\sum_e(t) = \sum_e(t-1) + g(\sum_e(t-1), \mu_e(t-1), x^{new}, x^{old}, N_{win}) \quad (13)$$

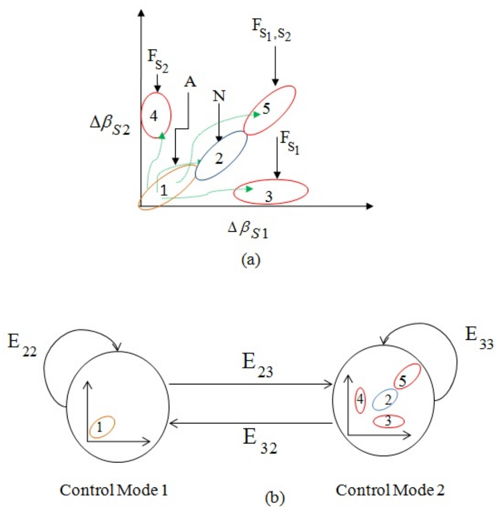


Figure 16: (a) Sensor decision space. (b) Control modes 1 and 2 modeled by a finite state automaton.

where x^{new} and x^{old} are respectively, the newest and the oldest arrived pattern in the time window N_{win} .

Initial off-line modeling allows the construction of initial classes that characterize knowledge from historical data. The historical data are usually sensor data that are saved. AuDyC is used to initialize the parameters of classes that will be dynamically updated. Knowledge of failure modes given from (labeled) historical data can help building a classification scheme for fault diagnosis. However, in reality, these data are hard to obtain.

In this work, we suppose that only data corresponding to normal operating conditions (normal classes) are known in advance. The training of the process by applying AuDyC is made based on features that are extracted from historical sensor data once finished; the class corresponding to normal operating conditions is retained. We denote this class by $C_N = (\mu_N, \Sigma_N)$.

In on-line functioning, the parameters of C_N are dynamically updated by AuDyC for each new pattern arrived in control mode 2. This yields changes in the class parameters which continuously reflect the distribution of the newest arriving patterns. We denote by $C_e = (\mu_e, \Sigma_e)$ the evolving classes in feature space. We have $C_e(t = 0) = (\mu_e, \Sigma_e) = C_N$.

In control mode 1 of pitch system, pitch sensor normal and faulty behaviors cannot be distinguished. Thus, in the proposed approach, the decisions about the status (normal/faulty) of patterns located in this region are delayed. Therefore in this case, the classifier will not be updated in order to avoid integrating in the drift time window useless patterns. In order to detect the drift as soon as possible, AuDyC updates the classes parameters by using a window that contains only the patterns belonging to control mode 2. AuDyC is dynamic by nature in the sense that it continuously updates the parameters of the classes as new patterns arrive.

6.3 Pattern decision analysis

When a new pattern is classified in the ambiguity class (A), in sensor feature space, assigning it to normal or failure operating conditions is a risky decision since normal and failure classes are overlapped in this region of the feature space. In order to reduce this risk, the decision about the status (normal or faulty) of any pattern classified in this region is delayed by assigning the label (A) (ambiguity decision). Then, this ambiguity can be removed by analyzing the past and future decisions of this pattern. The analysis of the pattern decision sequence is achieved by using a set of decision rules allowing assigning to ambiguity patterns label (N) or label (F) (normal or faulty) as follows. Let us suppose that $X_A = \{x_t, x_{t+1}, \ldots, x_{t+n}\}$ is a set of patterns associated with decision (A). Let x_{t-1} be the previous pattern arrived just before x_t. Let $D(x_{t-1}) \in \{A, N, F_i\}$ be the decision of this pattern. Let x_{t+n+1} the pattern arrived just after x_{t+n}. Let $D(x_{t+n+1}) \in \{A, N, F_i\}$ be the decision for this pattern. Then, the decision can be updated as follows:

$$D(x_{t-1}) = N \wedge D(x_{t+n+1}) = N \Rightarrow D(x) = N, \forall x \in X_A \tag{14}$$

$$D(x_{t-1}) = F \wedge D(x_{t+n+1}) = F \Rightarrow D(x) = F, \forall x \in X_A \tag{15}$$

$$D(x_{t-1}) = N \wedge D(x_{t+n+1}) = F \Rightarrow D(x) = A, \forall x \in X_A \tag{16}$$

$$D(x_{t-1}) = F \wedge D(x_{t+n+1}) = N \Rightarrow D(x) = A, \forall x \in X_A \tag{17}$$

Where $\wedge$ refers to And logical operation.

Rule 16 signifies that the fault has occurred somewhere in control mode 1 where its consequences on the pitch system dynamical behavior can be observed. Rule 17 indicates that the failure has disappeared in the control mode 1 either because of maintenance actions or because the fault is intermittent.

6.4 Drift monitoring and interpretation

The key problem of drift monitoring is to distinguish between variations due to stochastic perturbations and variations caused by unexpected changes in a system's state. If the sequence of observations is noisy, it may contain some inconsistent observations or measurements errors (outliers) that are random and may never appear again. Therefore, it is reasonable to monitor a system and to process observations within time windows in order to average and reduce the noise influence. Moreover, the information about possible structural changes within time windows can be interpreted and processed more easily. As a result, a more reliable classifier update can be achieved by monitoring within time windows. The latter must include enough of patterns representing the drift.

To distinguish the useful patterns, the pitch sensor dynamics are represented by two different control modes. In the control mode 2, the degradation consequences of pitch sensor can be observed. Therefore, all patterns in this mode are useful to be analyzed and to be included in the drift time window. In the control mode 1, the degradation consequences are masked. Patterns representing normal operating conditions cannot be distinguished from patterns representing pitch sensor degradations. Therefore in this case, no decision (normal/drift) will be taken in order to avoid integrating in the drift time window useless patterns.

The proposed scheme makes use of classes parameters (Mean, Variance-covariance matrix) which are dynamically updated at each time but only with the patterns belonging to control mode 2. Drift indicators are defined based on these parameters and the detection of faults inception will be made based on their values. We define two drift indicators $I_{h1}(x), I_{h2}(x)$ as follows:

$$I_{h_1}(x) = d_{Mah}(C_N, \mu_e) \quad (18)$$

$$I_{h_2}(x) = d_E(\mu_N, \mu_e) \quad (19)$$

Where d_{Mah} and d_E are, respectively, the Mahalanobis and Euclidean metrics.

Euclidean metric computes the distance between the center μ_n of the normal class C_N and the center μ_e of evolving class C_e; on the other side Mahalanobis metric computes the distance between the normal class C_N and the evolving class center μ_e. Therefore, these two distances are calculated as follows:

$$d_{Mah}(C_N, \mu_e) = \sqrt{(\mu_N - \mu_e) \quad \Sigma_N^{-1} \quad (\mu_N - \mu_e)^T} \quad (20)$$

$$d_E(\mu_N, \mu_e) = \sqrt{(\mu_N - \mu_e) \times (\mu_N - \mu_e)^T} \quad (21)$$

The drift is detected when the Mahalanobis indicator $I_{h_1}(x)$, defined by Equation 18, exceeds a certain threshold th_d:

$$I_{\text{h1}}(x) > th_d \Rightarrow \text{drift is detected} \quad (22)$$

After the drift detection, the drift is confirmed when Euclidean indicator $I_{h2}(x)$ defined by Equation 19, exceeds th_d as follows:

$$I_{\text{h2}}(x) > th_d \Rightarrow \text{drift is confirmed} \quad (23)$$

The selection of th_d is motivated statically by taking three σ (standard deviations) of the data in the normal operating conditions.

In the case of pitch sensor faults, three scenarios may appear in the sensor feature space: fault impacting sensor 1 (β_{m1}), fault impacting sensor 2 (β_{m2}) or fault impacting both sensors (β_{m1} and β_{m2}) at the same time. The direction of the evolving class in the sensor feature space depends on which of these scenarios happened. Therefore, for sensor fault isolation, we use a drift direction indicator in order to monitor the direction of the evolving class. This will allow to determine which of these three scenarios happened and hence to isolate the abnormal drift source. When drift occurs, the evolving class will migrate from normal operating condition to failure. The direction indicator Dr and direction isolation DI are used to isolate the sensor which caused the drift-like fault. The idea is to consider the angle θ_1 respectively θ_2, between the vector μ_e relating the center of the evolving class and the origin of the feature space, and the vector μ_{e1} respectively μ_{e2} relating the origin with the projection of the center of the evolving class according to feature 1 respectively feature 2, of the feature space. These angles define the movement direction of the evolving class.

In order to calculate θ_1 and θ_2, the scalar products between $\overrightarrow{\mu_{e1}}$ and $\overrightarrow{\mu_e}$ and between $\overrightarrow{\mu_{e2}}$ and $\overrightarrow{\mu_e}$ are calculated as follows:

$$\overrightarrow{\mu_e}(x) \cdot \overrightarrow{\mu_{e1}}(x) = \|\mu_e(x)\| \cdot \|\mu_{e1}(x)\| \cdot \cos\theta_1 \quad (24)$$

$$\overrightarrow{\mu_e}(x) \cdot \overrightarrow{\mu_{e2}}(x) = \|\mu_e(x)\| \cdot \|\mu_{e2}(x)\| \cdot \cos\theta_2 \quad (25)$$

If the drift is detected and confirmed by the two drift indicators $I_{h1}(x)$ and $I_{h2}(x)$, then the drift isolation (to determine if sensor 1 or sensor 2 or both is the source of this drift) is achieved as follows:

$$\text{If } Dr = \theta_1 - \theta_2 > th_a \text{ and } \theta_1 > \theta_2 \Rightarrow DI = 1 : \\ \text{fault in sensor 1 } (\beta_{m1}) (26)$$

$$\text{If } Dr = \theta_1 - \theta_2 > th_a \text{ and } \theta_2 < \theta_1 \Rightarrow DI = 2 : \\ \text{fault in sensor 2 } (\beta_{m2}) \ (27)$$

$$\text{If } Dr = \theta_1 - \theta_2 < th_a \Rightarrow DI = 3 : \\ \text{fault in both sensors } (\beta_{m1} \text{and} \beta_{m2}) \ (28)$$

where th_a is the angle threshold. th_a is defined according to the variation of patterns within the normal class C_N. Therefore, th_a is determined experimentally using the patterns belonging to C_N.

The interpretation step aims at interpreting the detected changes within the classifier parameters and structure. This interpretation is then used as a prediction about the tendency of the future development of the WT current situation. This prediction is useful to formulate a control or maintenance action.

7 Experimentation and obtained results

The failures of pitch sensors are caused by a continuous degradation of its performance over time. This degradation can be seen as a continuous drift of the normal operating conditions characteristics (normal class) of the pitch sensor. Detecting and following this drift can help to predict the occurrence of the pitch sensor failures. The two monitoring indicators defined by Equation 18 and Equation 19 are used to detect and to confirm this drift for the twenty-seven scenarios of simple and multiple drift-like fault in pitch sensors are defined in section 2.

7.1 Simple drift-like fault in sensor β_{m1}

Figure 18 and Figure 19 represent, respectively, first and second residuals used in the pitch sensor feature space in presence of an abnormal drift in pitch sensor 1, β_{m1}. We can see in the case of an abnormal drift in pitch sensor 1, β_{m1}, that only residual $\Delta\beta_{s1}$ is impacted, while residual $\Delta\beta_{s2}$ has similar behavior as the one without abnormal drift in β_{m1}.

Table 4 show the values of the drift indicators $I_{h_1}(x)$ and $I_{h_2}(x)$ for the nine defined drift-like fault scenarios. These values represent the required time (starting from the drift beginning) to detect and confirm the drift occurrence. Thus, they can be used as an evaluation criterion to measure the time delay to detect a drift before its end.

Figures 20 and 21 show the obtained results using the two drift detection indicators $I_{h_1}(x)$ and $I_{h_2}(x)$, for simple drift-like fault in pitch sensor β_{m1}. The degradation is observed when the pitch actuator operate in control mode 2, the drift like fault in pitch sensor is successfully detected by

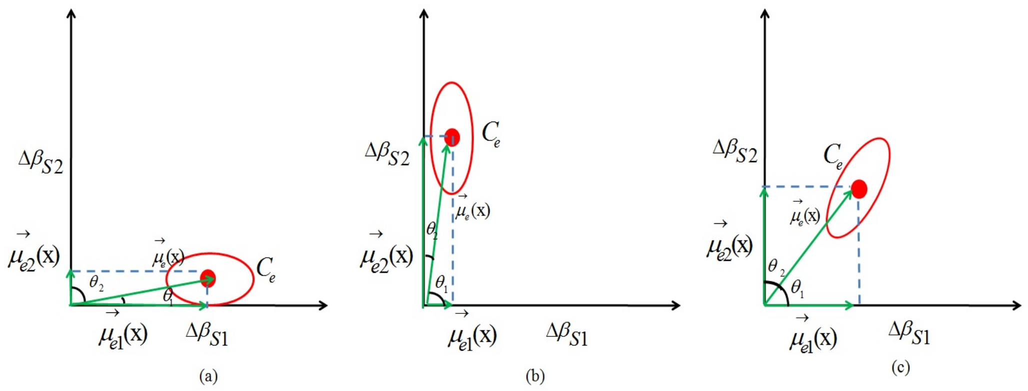


Figure 17: Drift direction angles in the pitch sensor feature space in the case of (a) simple drift-like fault in pitch sensor 1 (β_{m1}), (b) simple drift-like fault in pitch sensor 2 (β_{m2}), (c) multiple drift-like fault in both pitch sensors (β_{m1}) and (β_{m2}).

Fault N	Drift speed	I_{h1}	I_{h2}	Period
F_{4h}	30s(High) (High)	5.25s	11.00s	2500s -2730s
F_{4m}	60s(Medium) (Medium)	8.60s	18.70s	-2760s -2760s
F_{4s}	90s (Slow)	14s	26.30s	2500s -2790s
F_{5h}	30s	6.90s	13.30s	2600s -2830s
F_{5m}	60s	11.50s	20.20s	2600s -2860s
F_{5s}	90s	14.25s	27.10s	2600s -2890s
F_{6h}	30s	6.05s	11.90s	2700s -2930s
F_{6m}	60s	12.60s	23.50s	2700s -2960s
F_{6s}	90s	15.10s	29.40s	2700s -2990s

Table 4: Results of simple drift-like fault detection and confirmation in pitch sensor 1 (β_{m1}), for the nine drift scenarios.

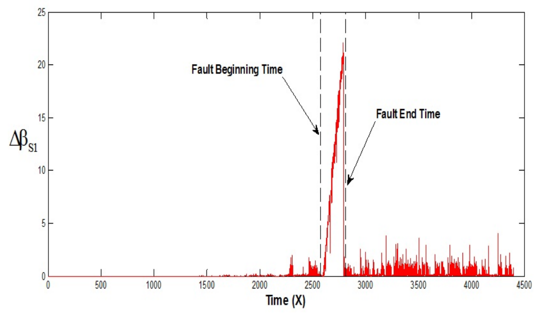


Figure 18: First residual used in the pitch sensor feature space in the case of the simple drift-like fault in pitch sensor 1 (β_{m1}).

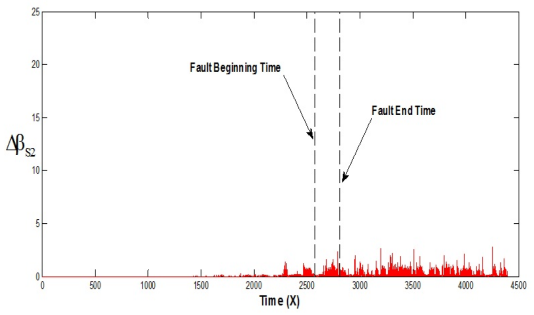


Figure 19: Second residual used in the pitch sensor feature space in the case of the simple drift-like fault in pitch sensor 1 (β_{m1}).

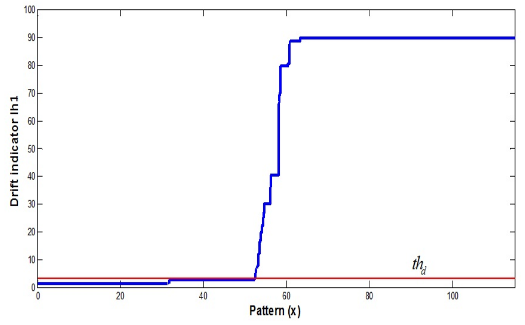


Figure 20: Drift indicator $I_{h_1}(x)$ based on Mahalanobis distance of the simple drift-like fault in pitch sensor 1 (β_{m1}).

both indicator $I_{h_1}(x)$ and $I_{h_2}(x)$, for all drift speeds (see Figure 20 and Figure 21).

The drift-like fault in pitch sensor 1 (β_{m1}), is detected in early stage before the end of this drift (arriving to the failure mode due to drift fault in pitch sensor). As an example, in the case of a drift of slow speed (F6s) (see Table 4), the

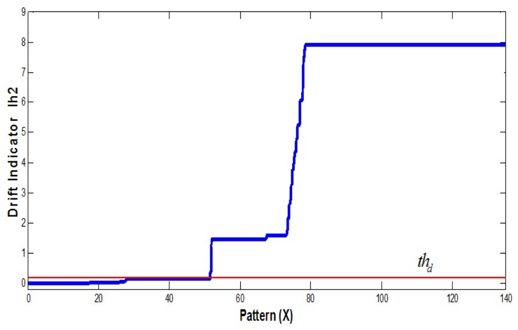


Figure 21: Drift indicator $I_{h_2}(x)$ based on Euclidean distance of the simple drift-like fault in pitch sensor 1 (β_{m1}).

pitch sensor reaches the failure mode resulting from a drift-like fault in λ_1 (degradation in λ_1) after 90 seconds of the beginning of the drift. In the proposed approach, this drift is detected 15.10 seconds and confirmed 29.40 seconds after its beginning. Therefore, the drift like fault in pitch sensor is confirmed 60 seconds before its end. This enables to achieve an early fault diagnosis and therefore helps the human operators of supervision to take efficiently the right actions.

Figure 22 and Figure 23 represent, respectively, evolving class angle and the direction indicator of the pitch sensor fault. These figures show the obtained results in presence of simple drift-like fault in pitch sensor 1, based on Figure 22 and Figure 23 the sensor 1 (β_{m1}), fault is successfully isolated by the direction indicator. Indeed, the direction angle shows that the evolving class exceeds the angle threshold (see Figure 17.a). Based on Equation 26, the drift-like fault in sensor 1 (β_{m1}), is isolated (see Figure 29).

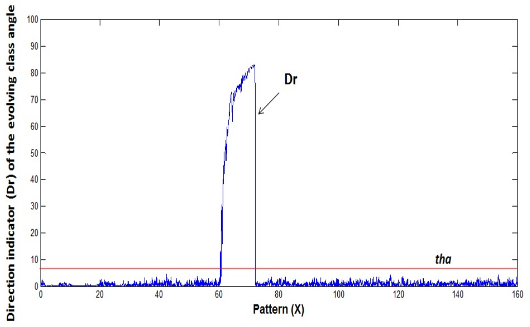


Figure 22: Direction indicator Dr of the evolving class angle of the simple drift-like fault in pitch sensor 1 (β_{m1}).

7.2 Simple drift-like fault in sensor β_{m2}

Figure 24 and Figure 25 represent, respectively, first and second residuals used in the pitch sensor feature space in presence of an abnormal drift in pitch sensor sensor 2, β_{m2}. We can see in the case of an abnormal drift in pitch sensor 2, β_{m2}, that only residual $\Delta\beta_{s2}$ is impacted, while residual $\Delta\beta_{s1}$ has similar behavior as the one without abnormal drift in β_{m2}.

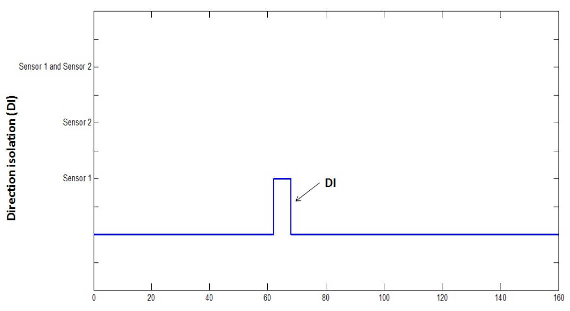


Figure 23: Direction isolation DI of the simple drift-like fault in pitch sensor 1 (β_{m1}).

Table 5 show the values of the drift indicators $I_{h_1}(x)$ and $I_{h_2}(x)$ for the nine defined drift-like fault scenarios. These values represent the required time (starting from the drift beginning) to detect and confirm the drift occurrence. Thus, they can be used as an evaluation criterion to measure the time delay to detect a drift before its end.

Fault N	Drift speed	I_{h1}	I_{h2}	Period
F_{7h}	30s (High)	6.07s	12.15s	2800s -3030s
F_{7m}	60s (Medium)	8.90s	19.05s	2800s -3060s
F_{7s}	90s (Slow)	14.20s	27s	2800s -3090s
F_{8h}	30s	5.70s	11.80s	2900s -3130s
F_{8m}	60s	8.25s	18.40s	2900s -3160s
F_{8s}	90s	13.70s	26.18s	2900s -3190s
F_{9h}	30s	6.90s	12.70s	3000s -3230s
F_{9m}	60s	9s	20.30s	3000s 3260s
F_{9s}	90s	14.90s	28.10s	3000s 3290s

Table 5: Results of simple drift-like fault detection and confirmation in pitch sensor 2(β_{m2}), for the nine drift scenarios.

Figures 26 and 27 show the obtained results using the two drift detection indicators $I_{h_1}(x)$ and $I_{h_2}(x)$, for simple drift-like fault in pitch sensor 2 (β_{m2}). The degradation is observed when the pitch actuator operate in control mode 2, the drift-like fault in pitch sensor 2 is successfully detected by both indicators $I_{h_1}(x)$ and $I_{h_2}(x)$ for all drift speeds (see Figure 26 and Figure 27).

The drift-like fault in pitch sensor 2 (β_{m2}), is detected in early stage before the end of this drift (arriving to the failure mode due to drift fault in pitch sensor). As an example, in the case of a drift of slow speed (F9s) (see Table 5), the pitch sensor reaches the failure mode resulting from a drift-like fault in λ_2 (degradation in λ_2) after 90 seconds of the beginning of the drift. In the proposed approach, this drift is detected 14.90 seconds and confirmed 28.10 seconds after

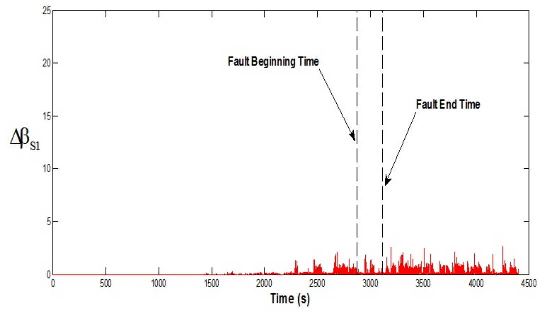


Figure 24: First residual used in the pitch sensor feature space in the case of the simple drift-like fault in pitch sensor 2 (β_{m2}).

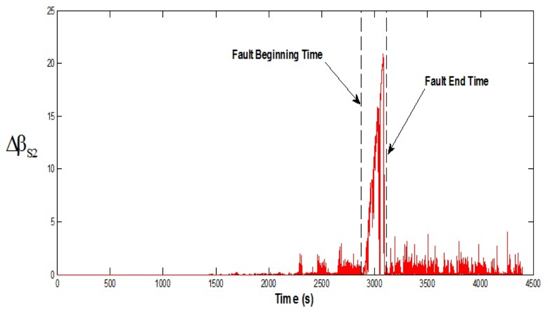


Figure 25: Second residual used in the pitch sensor feature space in the case of the simple drift-like fault in pitch sensor 2 (β_{m2}).

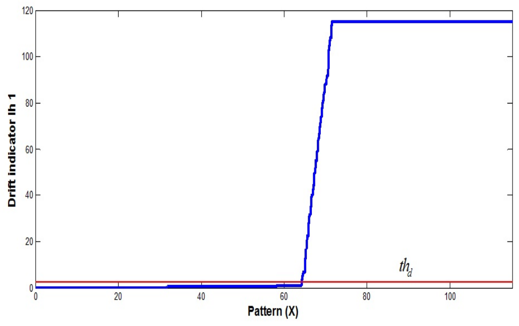


Figure 26: Drift indicator $I_{h_1}(x)$ based on Mahalanobis distance of the simple drift-like fault in pitch sensor 2 (β_{m2}).

its beginning. Therefore, the drift like fault in pitch sensor is confirmed 60 seconds before its end. This enables to achieve an early fault diagnosis and therefore helps the human operators of supervision to take efficiently the right actions.

For the drift isolation, Figure 28 and Figure 29 are used. They represent, respectively, evolving class angle and the direction indicator of the pitch sensor fault. These figures show the obtained results in presence of simple drift-like fault in pitch sensor 2, based on Figure 28 and Figure 29 the sensor 2 (β_{m2}), fault is successfully isolated by the direction indicator. Indeed, the direction angle shows that the evolving class exceeds the angle threshold (see Figure 17.b). Based on Equation 27, the drift-like fault in sensor 2 (β_{m2}), is isolated (see Figure 29).

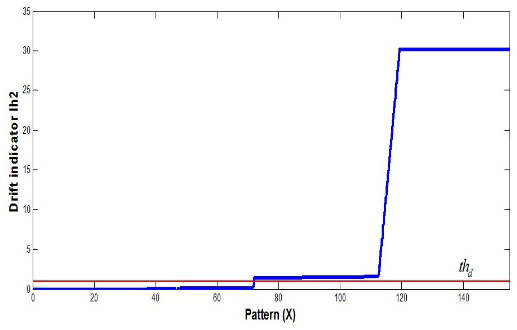


Figure 27: Drift indicator $I_{h_2}(x)$ based on Euclidean distance of the simple drift-like fault in pitch sensor 2 (β_{m2}).

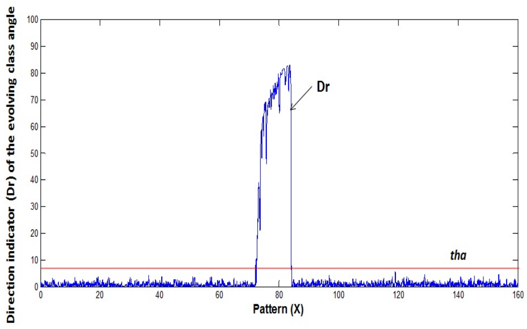


Figure 28: Direction indicator Dr of the evolving class angle of the simple drift-like fault in pitch sensor 2 (β_{m2}).

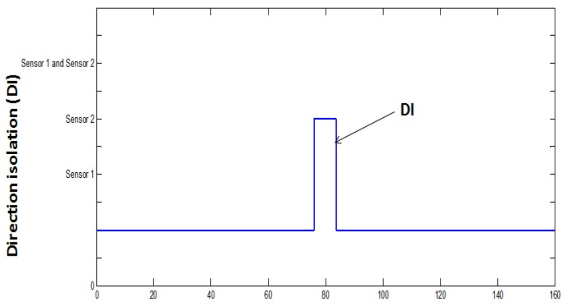


Figure 29: Direction isolation DI of the simple drift-like fault in pitch sensor 2 (β_{m2}).

7.3 Multiple drift-like fault in sensors β_{m1} and β_{m2}

Figure 30 and Figure 31 represent, respectively, first and second residuals used in the pitch sensor feature space in presence of an abnormal drift in both pitch sensor β_{m1} and β_{m2} at the same time. We can see that both residual $\Delta\beta_{s1}$ and $\Delta\beta_{s2}$ are impacted by the occurrence of the abnormal drift in β_{m1} and β_{m2}.

Table 6 show the values of the drift indicators $I_{h_1}(x)$ and $I_{h_2}(x)$ for the nine defined drift-like fault scenarios. These values represent the required time (starting from the drift beginning) to detect and confirm the drift occurrence. Thus, they can be used as an evaluation criterion to measure the time delay to detect a drift before its end.

Fault N	Drift speed	I_{h1}	I_{h2}	Period
F_{10h}	30s (High)	5.04s	10.9s	3100s -3330s
F_{10m}	60s (Medium)	9s	19.04s	3100s -3360s
F_{10s}	90s (Slow)	13.68s	26.23s	3100s -3390s
F_{11h}	30s	6.55s	15.50s	3200s -3430s
F_{11m}	60s	10.05s	19.30s	3200s -3460s
F_{11s}	90s	13.80s	27.50s	3200s -3490s
F_{12h}	30s	7.10s	16.10s	3300s -3530s
F_{12m}	60s	9.55s	22.80s	3300s -3560s
F_{12s}	90s	14.70s	28.25s	3300s -3590s

Table 6: Results of multiple drift-like fault detection and confirmation in pitch sensor 1 (β_{m1}), and pitch sensor 2 (β_{m2}), for the nine drift scenarios.

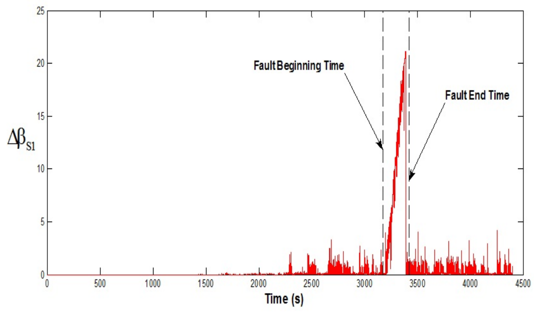


Figure 30: First residual used in the pitch sensor feature space in the case of the multiple drift-like fault in pitch sensor 1 (β_{m1}), and sensor 2 (β_{m2}).

Figures 32 and 33 show the obtained results using the two drift detection indicators $I_{h_1}(x)$ and $I_{h_2}(x)$, for multiple pitch sensor fault. The degradation is observed when the pitch actuator operate in control mode 2. The drift like

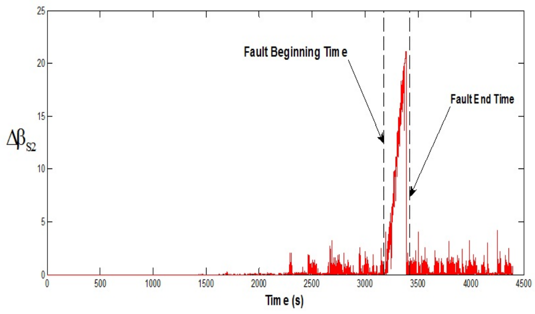


Figure 31: Second residual used in the pitch sensor feature space in the case of the multiple drift-like fault in pitch sensor 1 (β_{m1}), and sensor 2 (β_{m2}).

fault in pitch sensor is successfully detected by both indicator $I_{h_1}(x)$ and $I_{h_2}(x)$ for all drift speeds in both sensors (see Figure 32 and Figure 33).

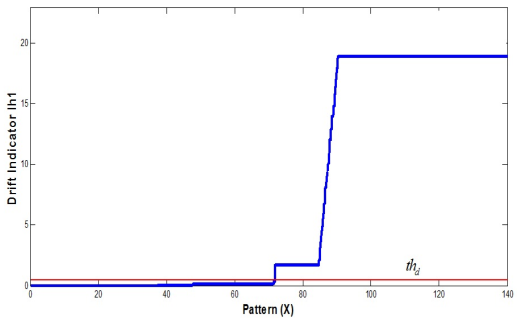


Figure 32: Drift indicator $I_{h_1}(x)$ based on Mahalanobis distance of the multiple drift-like fault in both pitch sensor 1 (β_{m1}), and sensor 2 (β_{m2}).

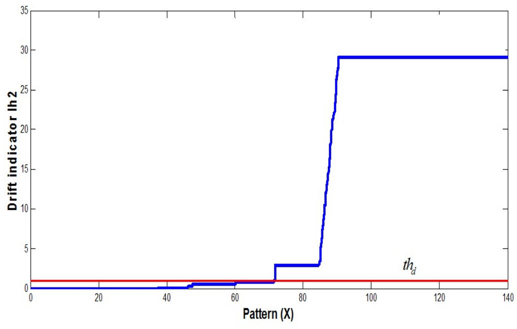


Figure 33: Drift indicator $I_{h_2}(x)$ based on Euclidean distance of the multiple drift-like fault in both pitch sensor 1 (β_{m1}), and sensor 2 (β_{m2}).

The multiple drift-like faults in pitch sensors are detected in early stage before the end of these drifts (arriving to the failure mode due to drift fault in both pitch sensors). As an example, in the case of a drift of slow speed (F12s) (see Table 6), the pitch sensors reache the failure mode resulting from a drift-like fault in λ_1 and λ_2 (degradation in λ_1 and λ_2) after 90 seconds of the beginning of the drift. In the proposed approach, this drift is detected 14.70 seconds and confirmed 28.25 seconds after its beginning. Therefore, the multiple drift-like fault in pitch sensor is confirmed 60 seconds before its end. This enables to achieve an early fault diagnosis and therefore helps the human operators of supervision to take efficiently the right actions.

For the drift isolation, Figure 34 and Figure 35 are used. They represent, respectively, evolving class angle and the direction indicator of the pitch sensor fault. These figures show the obtained results in presence of a multiple drift-like fault in both pitch sensors β_{m1} and β_{m2}, as we can see in Figure 34 and Figure 35 the fault is successfully isolated by the direction indicator. Indeed, the direction angle shows that the evolving class evolve within the axe of the normal class (see Figure 17.c). Based on Equation 27, the multiple drift-like isolation in both pitch sensors is isolated (see Figure 35).

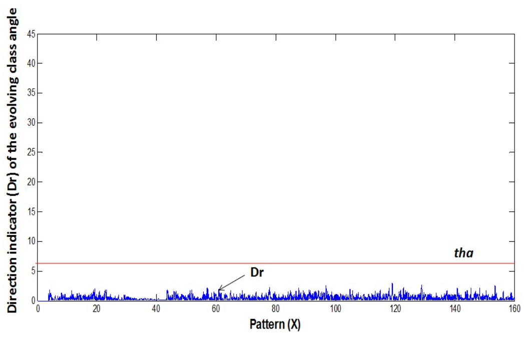


Figure 34: Direction indicator Dr of the evolving class angle of the multiple drift-like fault in both pitch sensor 1 (β_{m1}), and pitch sensor 2 (β_{m2}).

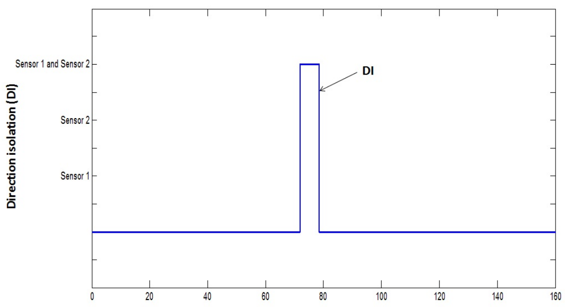


Figure 35: Direction isolation DI of the multiple drift-like fault in both pitch sensor 1 (β_{m1}), and sensor 2 (β_{m2}).

8 CONCLUSIONS

In this paper, an approach of condition monitoring and drift-like fault detection was developed. It is based on the use of a classifier able to achieve a reliable drift monitoring and early diagnosis of simple and multiple pitch sensors faults. This approach considers the system switching between several control modes. This approach based on the monitoring of the drift of the characteristics of classes representing the normal operating conditions of pitch system in each control mode. These characteristics are described by the mean and variance covariance matrix of these classes. They are monitored using two indicators in order to monitor and follow the drift. Both are defined based on the computation of the distance between the class representing normal operating conditions and the evolving class. The first indicator is based on the Mahalanobis distance and is used to detect the drift; while the second indicator is based on Euclidean distance and is used to confirm the drift. The drift indicators have detected successfully all drift scenarios of three speeds in early stage before the end of this drift for the case of simple and multiple drift-like faults in pitch system.

Future work will focus on the drift like fault of other wind turbine critical components as the generator and drive train as well as the use of other indicators to detect drifts of other types or natures.

References

[1] Bouthaina Abichou, Diana Flórez, Moamar Sayed-Mouchaweh, Houari Toubakh, Bruno François, and Nicolas Girard. Fault diagnosis methods for wind turbines health monitoring: a review. In *European Conference of the Prognostics and Health Management Society*, 2014.

[2] J.C. Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 1981.

[3] W. Chen, S. X. Ding, A. Haghani, A. Naik, A. Q. Khan, and S. Yin. Observer-based fdi schemes for wind turbine benchmark. pages 7073–7078, 2011.

[4] MA Djeziri, H Toubakh, and M Ouladsine. Fault prognosis based on fault reconstruction: Application to a mechatronic system. In *Systems and Control (ICSC), 2013 3rd International Conference on*, pages 383–388. IEEE, 2013.

[5] D.Kolev, P.P. Angelov, G. Markarian, M.Suvorov, and S.Lysanov. Arfa: Automated real-time flight data analysis using evolving clustering, classifiers and recursive density estimation. In *Evolving and Adaptive Intelligent Systems (EAIS), 2013 IEEE Conference on*, pages 91–97, April 2013.

[6] J. M. P. PÃľrez M. Papaelias F. P. G. MÃąrquez, A. M. Tobias. Condition monitoring of wind turbines: Techniques and methods. *Renewable Energy*, 46:169–178, 2012.

[7] Fredrik Gustafsson and Fredrik Gustafsson. *Adaptive filtering and change detection*, volume 1. Wiley New York, 2000.

[8] R. Hallouzi. *Multiple-model based diagnosis for adaptive fault-tolerant control*. TU Delft, Delft University of Technology, 2008.

[9] A. Kusiak and W. Li. The prediction and diagnosis of wind turbine faults. *Renewable Energy*, 36:16–23, 2011.

[10] A. Kusiak and A. Verma. A data-mining approach to monitoring wind turbines. *Sustainable Energy, IEEE Transactions on*, 3(1):150–157, Jan 2012.

[11] N. Laouti, S. Othman, M. Alamir, and N.Sheibat-Othman. Combination of model-based observer and support vector machines for fault detection of wind turbines. *International Journal of Automation and Computing*, 11:274–287, 2015.

[12] X. Luo and X. Huang. Fault diagnosis of wind turbine based on elmd and fcm. *The Open Mechanical Engineering Journal*, 8:716–720, year= 2014,.

[13] M.Traore, E. Duviella, and S. Lecoeuche. Comparison of two prognosis methods based on neuro fuzzy inference system and clustering neural network. In *7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, pages 91–97, 2009.

[14] P. F. Odgaard and J. Stoustrup. Unknown input observer based scheme for detecting faults in a wind turbine converter. pages 161–166, 2009.

[15] P.F. Odgaard, J. Stoustrup, and M. Kinnaert. Fault-tolerant control of wind turbines: A benchmark model. *Control Systems Technology, IEEE Transactions on*, 21(4):1168–1182, July 2013.

[16] Ahmet Arda Ozdemir, Peter Seiler, and Gary J Balas. Wind turbine fault detection using counter-based residual thresholding. *IFAC Proceedings Volumes*, 44(1):8289–8294, 2011.

[17] R. Precup, P. Angelov, B. S. Jales Costa, and M. Sayed-Mouchaweh. An overview on fault diagnosis and nature-inspired optimal control of industrial process applications. *Computers in Industry*, 2015.

[18] S. Simani, S. Farsoni, and P. Castaldi. Residual generator fuzzy identification for wind turbine benchmark fault diagnosis. *machines*, 2:275–298, 2014.

[19] Silvio Simani, Paolo Castaldi, and Marcello Bonfe. Hybrid model–based fault detection of wind turbine sensors. *IFAC Proceedings Volumes*, 44(1):7061–7066, 2011.

[20] S. Tabatabaeipour, P.F. Odgaard, T. Bak, and J. Stoustrup. Fault detection of wind turbines with uncertain parameters: a set-membership approach. *Energies*, 5(7):2424–2448, 2012.

[21] C. Tsai, C. Hsieh, and S. Huang. Enhancement of damage-detection of wind turbine blades via cwt-based approaches. *Energy Conversion, IEEE Transactions on*, 21(3):776–781, Sept 2006.

[22] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin. A review of process fault detection and diagnosis part iii: Process history based methods. *Computers and Chemical Engineering*, 27:327–346, 2003.

[23] S.Zhao R.M. Ferrari M.M. Polycarpou X.Zhang, Q. Zhang and T. Parisini. Fault detection and isolation of the wind turbine benchmark: An estimation-based approach. In *Proceedings of IFAC world congress*, volume 2, pages 8295–8300, 2011.

[24] W. Yang, R. Court, and J. Jiang. Wind turbine condition monitoring by the approach of scada data analysis. *Renewable Energy*, 53:365–376, 2013.

[25] X. Youa and W. Zhangb. Fault diagnosis of frequency converter in wind power system based on som neural network. *Procedia Engineering*, 29:3132âĂŞ3136, 2012.

[26] Y. Zhi-Ling, W. Bin, D. Xing-Hui, and L. Hao. Expert system of fault diagnosis for gear box in wind turbine. *Systems Engineering Procedia*, 4:189–195, 2012.

Self-Adaptive Ensemble Classifier for Handling Complex Concept Drift

Imen Khamassi[1], Moamar Sayed-Mouchaweh[2]

1. Université de Tunis, Institut Supérieur de Gestion de Tunis, Tunisia
imen.khamassi@isg.rnu.tn
2. Ecole des Mines Douai, France
moamar.sayed-mouchaweh@mines-douai.fr

Abstract. In increasing number of real world applications, data are presented as streams that may evolve over time and this is known by *concept drift*. Handling concept drift through ensemble classifiers has received a great interest in last decades. The success of these ensemble methods relies on their *diversity*. Accordingly, various diversity techniques can be used like *block-based data*, *weighting-data* or *filtering-data*. Each of these diversity techniques is efficient to handle certain characteristics of drift. However, when the drift is complex, they fail to efficiently handle it. *Complex drifts* may present a mixture of several characteristics (speed, severity, influence zones in the feature space, etc) which may vary over time. In this case, drift handling is more complicated and requires new detection and updating tools. For this purpose, a new ensemble approach, namely EnsembleEDIST2, is presented. It combines the three diversity techniques in order to take benefit from their advantages and outperform their limits. Additionally, it makes use of EDIST2, as drift detection mechanism, in order to monitor the ensemble's performance and detect changes. EnsembleEDIST2 was tested through different scenarios of complex drift generated from synthetic and real datasets. This diversity combination allows EnsembleEDIST2 to outperform similar ensemble approaches in term of accuracy rate, and present stable behaviors in handling different scenarios of complex drift.

Keywords:Ensemble Classifier, Diversity techniques, Complex Concept Drift, Adaptive Learning, Evolving Data Stream, Change Detection

1 Introduction

Learning from evolving data stream has received a great attention. It addresses the non-stationarity of data over time, which is known by *concept drift*. The term *concept* refers to data distribution, represented by the joint distribution $p(x, y)$, where x represents the $n - dimensional$ feature vector and y represents its class label. The term *concept drift* refers to a change in the underlying distribution of new incoming data. For example, in intrusion detection application, the behavior of an intruder may evolve in order to confuse the system protection rules. Hence, it is essential to consider these changes for updating the system in order to preserve its performance.

Ensemble classifiers appear to be promising approaches for tracking evolving data streams. The success of the ensemble methods, according to single classifier, relies on their *diversity* [17] [22] [21]. *Diversity* can be achieved according to three main strategies [15]: *block-based data*, *weighting-data* or *filtering-data*. In *block-based ensembles* [5], [16], [20], the training set is presented as blocks or chunks of data at a time. Generally, these blocks are of equal size and the evaluation of base learners is done when all instances from a new block are available. In *weighting-data ensembles* [3] [4] [18] [13], the instances are weighted according to some weighting process. For example in Online Bagging [19], the weighting process is based on re-using instances for training individual learners. Finally, *filtering-data ensembles* [1] are based on selecting data from the training set according to a specific criterion, for example similarity in feature space.

In many real-life applications, the concept drift may be *complex* in the sense that it presents time-varying characteristics. For instance, a drift can present different characteristics according to its speed (*abrupt or gradual*), nature (*continuous or probabilistic*) and severity (*local or global*). Accordingly, *complex drift* can present a mixture of all these characteristics over time. It is worth to underline that each characteristic presents its own challenges. Accordingly, a mixture of these different characteristics may accentuate the challenge issues and complicate the drift handling.

In this paper, the goal is to underline the complementarity of the diversity techniques (*block-based data, weighting-data* and *filtering-data*) for handling different scenarios of complex drift. For this purpose, a new ensemble approach, namely EnsembleEDIST2, is proposed. The intuition is to combine these three diversity techniques in order to efficiently handle different scenarios of complex drift. Firstly, EnsembleEDIST2 defines a data-block with variable size for updating the ensemble's members, thus it can avoid the problem of tuning off size of the data-block. Secondly, it defines a new filtering criterion for selecting the most representative data of the new concept. Thirdly, it applies a new weighting process in order to create diversified ensemble's members. Finally, it makes use of EDIST2 [14] [12], as drift detection mechanism, in order to monitor the ensemble's performance and detect changes.

EnsembleEDIST2 has been tested through different scenarios of complex drifts generated from synthetic and real datasets. This diversity combination allows EnsembleEDIST2 to outperform similar ensemble approaches in term of accuracy rate, and present a stable behavior in handling different scenarios of complex drift.

The remainder of the paper is organized as follows. In *Section II*, the challenges of complex concept drift are exposed. In *Section III*, the advantages and the limits of each diversity technique are studied. In *Section IV*, the proposed approach, namely EnsembleEDIST2, is detailed. *Section V*, the experimental setup and the obtained results are presented. Finally, in *Section VI*, the conclusion and some future research directions are exposed.

2 Complex Concept Drift

In many real-life applications, the concept drift may be *complex* in the sense that it presents time-varying characteristics. Let us take the example of a drift with three different characteristics according to its speed *(gradual or abrupt)*, nature *(continuous or probabilistic)* and severity *(local or global)*. It is worth to underline that each characteristic presents its own challenges. Accordingly, a mixture of these different characteristics may accentuate the challenge issues and complicate the drift handling.

For instance, we can consider the drift depicted in Fig.1 as complex drift as it simulates a Gradual Continuous Local Drift, in the sense that the hyperplane class boundary is gradually rotating during the drifting phase and continuously presenting changes with each instance in local regions. Namely, the time until this complex drift is detected can be arbitrarily long. This is due to the rarity of data source representing the drift, which in turn makes it difficult to confirm the presence of drift. Moreover, in some cases, this drift can be considered as noise by confusion, which makes the model unstable. Hence, to overcome the instability, the model has to *(i)* effectively differentiate between local changes and noises, and *(ii)* deal with the scarcity of instances that represent the drift in order to effectively update the learner.

Another interesting complex drift represents the Gradual Continuous Global Drift (see Fig.2). During this drift, the concept is gradually changing and continuously presenting modifications with each instance. Namely, during the transition phase, the drift evolves and presents several intermediate concepts until the emergence of the final concept (see Fig.2.b). Hence, the challenging issue is to efficiently decide the end time of the old concept and detect the start time of the new concept. The objective is to update the learner with the data that represent the final concept (see Fig.2.c) and not with data collected during the concept evolution (see Fig.2.b). Moreover, this drift is considered as global because it is affecting all the instances of the drifting class. Namely, handling this complex drift is also challenging, because the performance's decrease of the learner is more pronounced than the other types of drifts.

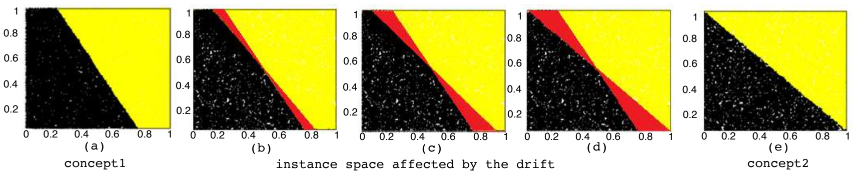


Fig. 1. Gradual Continuous Local Drift:**a** concept1, **b-d** instance space affected by the drift and **e** concept2

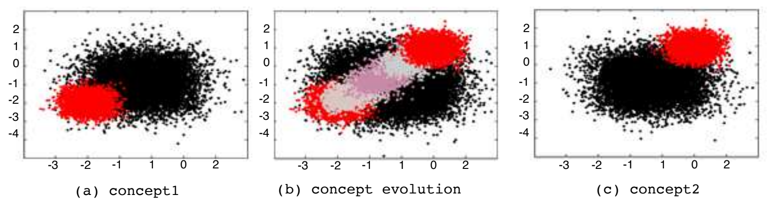


Fig. 2. Gradual Continuous Global Drift: **a** concept1, **b** concept evolution and **c** concept2

3 Related work

The *diversity* [15] among the ensemble can be fulfilled by applying various techniques such as: *block-based data*, *weighting-data* or *filtering data*, in order to differently train base learners (see Fig.3). Accordingly, the objective in this investigation is to highlight the advantages and drawbacks of each diversity techniques in handling complex drift (see Table 1).

3.1 Block-based Technique

According to the *block-based technique*, the training set is presented as blocks or chunks of data at a time. Generally, these blocks are of equal size and the construction, evaluation, or updating of base learners is done when all instances from a new block are available. Very often, ensemble learners periodically evaluate their components and substitute the weakest one with a new (candidate) learner after each data block [20] [16] [6]. This technique preserves the adaptability of the ensemble in such way that learners, which were trained in recent blocks, are the most suitable for representing the current concept.

The block-based ensembles are suitable for handling gradual drifts. Generally, during these drifts, the change between consecutive data blocks is not quite pronounced; thus, it can be only noticeable in long period. The interesting point in the block-based ensembles is that they can enclose different learners that are trained in different period of time. Hence, by aggregating the outputs of these base classifiers, the ensemble can offer accurate reactions to such gradual drifts.

In contrast, the main drawback of block-based ensembles is the difficulty of tuning off the block size to offer a compromise between fast reactions to drifts and high accuracy. If the block size is too large, they may slowly react to abrupt drift; whereas small size can damage the performance of the ensemble in stable periods.

3.2 Weighting-data Technique

In this technique, the base learners are trained according to weighted instances from the training set. A popular instance weighting process is presented in the Online Bagging ensemble [19]. For ease of understanding, the weighting process is based on re-using instances for training individual classifiers. Namely, if we consider that each base classifier C_i is trained from a subset M_i from the global training set; then the $instance_i$ will be presented k times in M_i; where the weight k is drawn from a $Poisson(1)$ distribution.

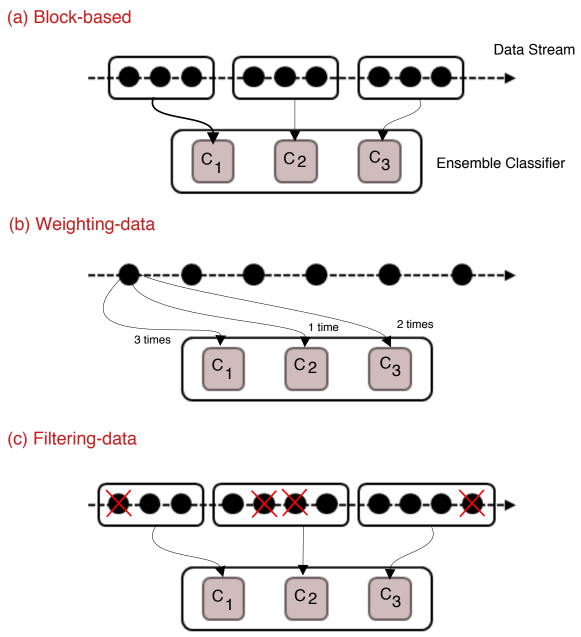


Fig. 3. Different diversity techniques among the ensemble

Online Bagging has inspired many researchers in the field of drift tracking [3] [17] [13]. This approach can be of great interest for:

- Class imbalance: where some classes are severely underrepresented in the dataset
- Local drift: where changes occur in only some regions of the instance space.

Generally, the weighting process intensifies the re-use of underrepresented class data and helps to deal with the scarcity of instances that represent the local drift. However, the instance duplication may impact the ability of the ensemble in handling global drift. During global drift, the change affects a large amount of data; thus when re-using data for constructing base classifiers, the performance's decrease is accentuated and the recovery from the drift may be delayed.

3.3 Filtering-data Technique

This technique is based on selecting data from the training set according to a specific criterion, for example similarity in the feature space. Such technique allows to select subsets of attributes that provide partitions of the training set containing maximally similar instances, i.e., instances belonging to the same regions of feature space. Thanks to this technique, base learners are trained according to different subspaces to get benefit from different characteristics of the overall feature space.

In contrast with conventional approaches which detect drift in the overall distribution without specifying which feature has changed, ensemble learners based on filtered data can exactly specify the drifting feature. This is a desired property for detecting novel class emergence or existing class fusion in unlabeled data. However, these approaches may present difficulty in handling local drifts if they do not define an efficient filtering criterion. It is worth to underline that during local drift, only some regions of the feature space are affected by the drift. Hence, only the base classifier which is trained on changing region is the most accurate to handle the drift. However, when aggregating the final decision of this classifier with the remained classifiers, trained from unchanged regions, the performance recovery may be delayed.

Table 1. Summary of the advantages (+) and drawbacks (-) of diversity techniques for handling complex drift

Complex Drift	Gradual Continuous		Gradual Probabilistic		Abrupt	
	Local	Global	Local	Global	Local	Global
Block-based	+	+	+	+	-	-
Weighting-data	+	-	+	-	+	-
Filtering-data	-	+	-	+	-	+

4 The proposed approach

The intuition behind EnsembleEDIST2 is to combine the three diversity techniques (*Block-based*, *Weighting-data* and *Filtering data*) in order to take benefit from their advantages and avoid their drawbacks.

The contributions of EnsembleEDIST2 for efficiently handling complex concept drifts are as follows, it:

- Explicitly handles drift through a drift detection method EDIST2 [14] (subSection4.1)
- Makes use of data-block with variable size for updating the ensemble's members (subSection4.2)

– Defines a new filtering criterion for selecting the most representative data of the new concept (subSection4.3)
– Applies a new weighting process in order to create diversified ensemble's members (subSection4.4)

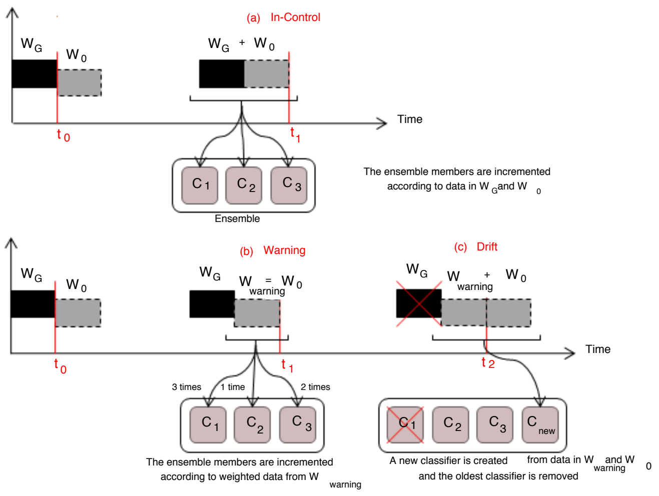

(a) In-Control
W_G
W_0
W_G + W_0
t_0
t_1
Time
C_1 C_2 C_3
Ensemble
The ensemble members are incremented according to data in W_G and W_0
(b) Warning
W_G
$W_{warning}$ = W_0
t_0
t_1
3 times
1 time
2 times
The ensemble members are incremented according to weighted data from $W_{warning}$
(c) Drift
W_G
$W_{warning}$ + W_0
t_2
Time
C_1 C_2 C_3 C_{new}
A new classifier is created from data in $W_{warning}$ and W_0 and the oldest classifier is removed

Fig. 4. EnsembleEDIST2's adapting process according to the three detection levels: (a) In-control, (b) Warning and (c) Drift

4.1 Drift monitoring process in EnsembleEDIST2

EnsembleEDIST2 is an ensemble classifier designed to explicitly handle drifts. It makes use of EDIST2 [14], as drift detection mechanism, in order to monitor the ensemble's performance and detect changes (see Fig4).

EDIST2 monitors the prediction feedback provided by the ensemble. More precisely, EDIST2 studies the distance between two consecutive errors of classification. Notice that the distance is represented by the number of instances between two consecutive errors of classification. Accordingly, when the data distribution becomes non-stationary, the ensemble will commit much more errors and the distance between these errors will decrease.

In EDIST2, the concept drift is tracked through two data windows, a 'global' one and a 'current' one. The global window W_G is a self-adaptive window which is continuously incremented if no drift occurs and decremented otherwise; and the current window W_0 which represents the batch of current collected instances.

In EDIST2, we want to estimate the error distance distribution of W_G and W_0 and make a comparison between the averages of their error distance distributions in order to check a difference. As stated before, a significant decrease in the error distance implies a change in the data distribution and suggests that the learning model is no longer appropriate.

EDIST2 makes use of a statistical hypothesis test in order to compare W_G and W_0 error distance distributions and check whether the averages differ by more than the threshold ϵ. It is worth underlining that there is no *a priori* definition of the threshold ϵ, in the sense that it does not require any *a priori* adjusting related to the expected speed or severity of the change. ϵ is autonomously adapted according to a statistical hypothesis test (for more details please refer ti [14]).

The intuition behind EDIST2 is to monitor μ_d which represents difference between W_G and W_0 averages and accordingly three thresholds are defined:

– *In-Control level*: $\mu_d \leq \epsilon$; within this level, we confirm that there is no change between the two distributions, so we enlarge W_G by adding W_0 's instances. Accordingly, all the ensemble members are incremented according to data samples in W_G and W_0.
– *Warning level*: $\mu_d > \epsilon$; within this level, the instances are stored in an warning chunk $W_{warning}$. Accordingly, all the ensemble members are incremented according to weighted data from $W_{warning}$. (The weighting process will be explained in *subSection4.4*)
– *Drift level*: $\mu_d > \epsilon + \sigma_d$; within this level, the drift is confirmed and W_G is decremented by only containing the instances stored since the warning level,i.e., in $W_{warning}$. Additionally, a new base classifier is created from scratch and trained according to data samples in $W_{warning}$, then the oldest classifier is removed from the ensemble.

4.2 EnsembleEDIST2's diversity by variable-sized block technique

In EnsembleEDIST2, the size of data-block is not defined according to the number of instances, as it is the case of conventional *block-based ensembles*, but according to the number of errors committed during the learning process. More precisely, the data-block W_0, in EnsembleEDIST2, is constructed by collecting the instances that exist between N_0 errors.

As depicted in Fig.5 , when the drift is abrupt, the ensemble commits N_0 errors in short drifting time. However, when the drift is gradual, the ensemble commits N_0 errors in relatively longer drifting time. Hence, according to this strategy, the block size is variable and adjusted according to drift characteristics.

It is worth to underline that EnsembleEDIST2 can offer a compromise between fast reaction to abrupt drift and stable behavior regarding gradual drift. This is a desirable property for handling complex drift which may present different characteristics in the same time, and accordingly EnsembleEDIST2 can avoid the problem of tuning off the size of data-block as it is the case of most block-based approaches.

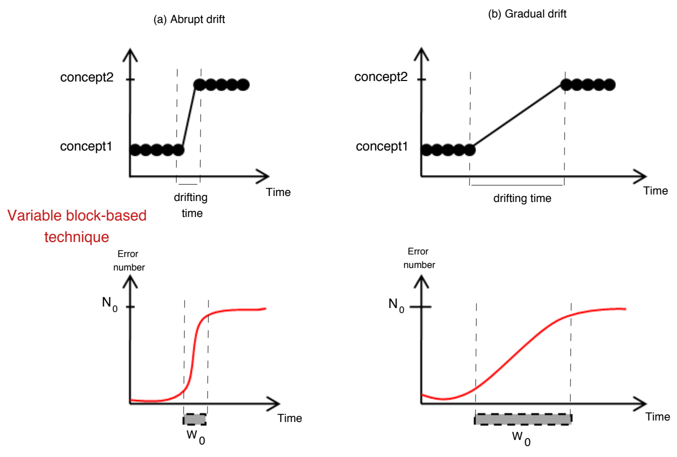


Fig. 5. Variable data-block technique in EnsembleEDIST2

4.3 EnsembleEDIST2's diversity by new filtering-data criterion

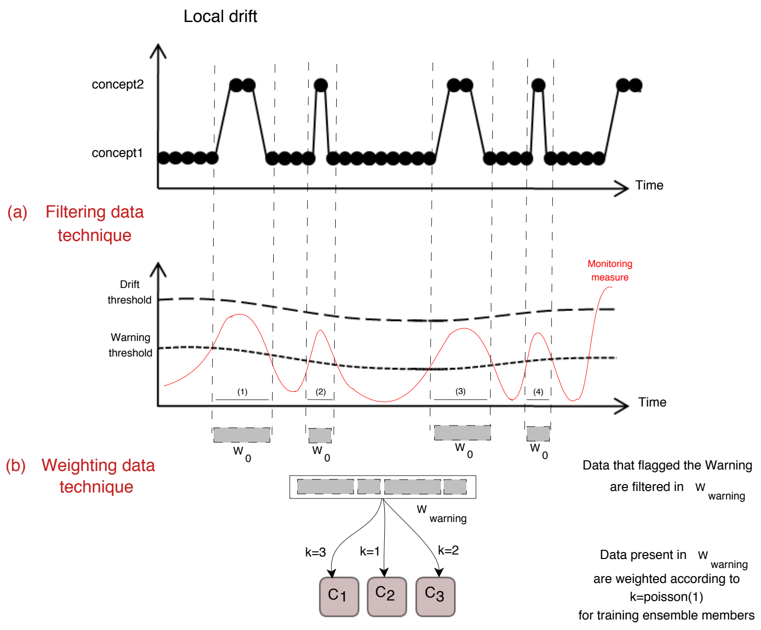
Differently from conventional filtering-data ensembles, which filter data according to similarity in the feature space, EnsembleEDIST2 defines a new filtering criterion. It filters the instances that trigger the warning level. More precisely, each time the ensemble reaches the warning level, the instances are gathered in a warning chunk $W_{warning}$ in order to re-use them for training the ensemble's members (see Fig.6.a). This is an interesting point when dealing with local drift because drifting data are scarce and not continuously provided. It is possible that a certain amount of drifting data can be found in zones (1), (2), (3) and (4) but not quite sufficient to reach the drift level. Accordingly, by considering these data for updating the ensemble's members, EnsembleEDIST2 can ensure a rapid recovery from local drift.

In contrast, conventional filtering-data ensembles are unable the define in which zone the drift has occurred, thus, they may update the ensemble's members with data filtered from unchanged feature space; which in turn may delay the performance correctness.

4.4 EnsembleEDIST2's diversity by new weighting-data process

The focus in EnsembleEDIST2 is to maximize the use of data present in $W_{warning}$ for accurately updating the ensemble. More precisely, the data in $W_{warning}$ are weighted according to the same weighting process used in Online bagging [19]. Namely, each $instance_i$ from $W_{warning}$ is re-used k times for training the base classifier C_i , where the weight k is drawn from a $Poisson(1)$ distribution (see Appendix7).

Generally, the weighting process in EnsembleEDIST2 offers twofold advantages. First, it intensifies the re-use of underrepresented class data and helps to deal with scarcity of instances that represent the local drift. Second, it permits faster recovery from global drift than conventional weighting-data ensembles. As it is known, during global drift, the change affects a large amount of data. Hence, differently from conventional weighting-data ensembles, which apply the weighting process to all the data sets; EnsembleEDIST2 only weights the instances present in $W_{warning}$ (see Fig.6.b). Accordingly, it can avoid to accentuate the decrease of the ensemble's performance during global drift, and ensure a fast recovery.



Local drift
concept2
concept1
Time
(a) Filtering data technique
Drift threshold
Warning threshold
Monitoring measure
(1)
(2)
(3)
(4)
Time
W0
W0
W0
W0
(b) Weighting data technique
Data that flagged the Warning are filtered in Wwarning
Wwarning
k=3
k=1
k=2
C1
C2
C3
Data present in Wwarning are weighted according to k=poisson(1) for training ensemble members

Fig. 6. (a) Filtering-data technique and (b) Weighting-data technique in EnsembleEDIST2

5 Experiments and performance analysis

5.1 Experimental evaluation

Synthetic Datasets In this investigation, we are studying six different scenarios of complex concept drift as depicted in Table 2 . All synthetic datasets contain $100,000$ instances and one concept drift where the starting and the ending time are predefined. For gradual drift, the drifting time lasts $30,000$ instances (it begins at t_{start}=40,000 and ends at $t_{end} = 70,000$). For abrupt drift, the drift occurs at $t = 50,000$.

Table 2. Different types of Complex Drift handled in this investigation

Complex Drift Characteristics			**Synthetic Datasets**
Speed	**Nature**	**Severity**	
Gradual	Continuous	Local	Hyperplane [10]
		Global	RBF [2]
	Probabilistic	Local	SEA Gradual [24]
		Global	STAGGER Gradual [23]
Abrupt		Local	SEA Abrupt [24]
		Global	STAGGER Abrupt [23]

Real Datasets

Electricity Dataset (48,312 instances, 8 attributes, 2 classes) is a real world dataset from the Australian New South Wales Electricity Market [9]. In this electricity market, the prices are not fixed and may be affected by demand and supply. The dataset covers a period of two years and the instances are recorded every half an hour. The classification task is to predict a rise (UP) or a fall (DOWN) in the electricity price. Three numerical features are used to define the feature space: the electricity demand in current region, the electricity demand in the adjacent regions and the schedule of electricity transfer between the two regions.

This dataset may present several scenarios of complex drift. For instance, a gradual continuous drift may occur when the users progressively change their consumption habits during a long time period. Likewise, an abrupt drift may occur when the electricity prices suddenly increase due to unexpected events (e.g., political crises or natural disasters). Moreover, the drift can be local if it impacts only one feature (e.g., the electricity demand in current region); or global if it impacts all the features.

Spam Dataset (9,324 instances, 500 attributes, 2 classes) is a real world dataset containing email messages from the Spam Assassin Collection Project [11]. The classification task is to predict if a mail is a spam or legitimate. The data set contains 20% of spam mailing. The feature space is defined by a set of numerical features such as the number of receptors, textual attributes describing the mail contain and sender characteristics...

This dataset may present several scenarios of complex drift. For instance, a gradual drift may occur when the user progressively changes his preferences. However, an abrupt drift may occur when the spammer rapidly changes the mail content to trick the spam filter rules. It is worth to underline that the drift can also be continuous when the spammer starts to change the spam content; but the filter continues to correctly detect them. In the other side, the drift can be probabilistic when the spammer starts to change the spam content; but the filter fails in detecting some of them.

Evaluation criteria When dealing with evolving data streams, the objective is to study the evolution of the EnsembleEDIST2 performance over time and see how quick the adaptation to drift is. According to Gama et al. [8] the prequential accuracy is a suitable metric to evaluate the learner performance in presence of concept drift. It proceeds as follows: each instance is firstly used for testing then for training. Hence, the accuracy is incrementally updated using the maximum available data; and the model is continuously tested on instances that it has not already seen (for more details please refer to [8]).

Parameter Settings All the tested approaches were implemented in the java programming language by extending the Massive Online Analysis (MOA) software [2]. MOA is an online learning framework for evolving data streams and supports a collection of machine learning methods.

For comparison, we have selected well known ensemble approaches according to each category:

– *Block-based ensemble*: AUE (Accuracy Updated Ensemble) [5], AWE (Accuracy Weighted Ensemble) [16] and LearnNSE [20] with block size equal to 500 instances.
– *Weighting-data ensemble*: LeveragingBag [3] and OzaBag [19]
– *Filtering-data ensemble*: LimAttClass [1]

For all these approaches, the ensemble's size was fixed to 10 and the Hoeffding Tree (HT) [7] was used as base learning algorithm.

It is worth to notice that EnsembleEDIST2 makes use of two parameters: N_0 which is the number of error in W_0 and m which is the number of base classifiers among the ensemble. In this investigation, we respectively set $N_0 = 30$ and $m = 3$ according to empirically studies done in *subSections* 5.2 and 5.2.

5.2 Comparative study and interpretation

Impact of N_0 on EnsembleEDIST2 performance EnsembleEDIST2 makes use of the parameter N_0 in order to define the minimum number of error occurred in W_0. Recall that W_0 represents the batch of current collected instances. This batch is constructed by collecting the instances that exist between N_0 errors.

It is interesting to study the impact of N_0 on the accuracy according to different scenarios of complex drift. For this purpose, we have done the following experiments: for each scenario of complex drift, the accuracy of EnsembleEDIST2 is presented by varying N_0 values (see Table 3).

Based on these results, we can conclude that the performance of EnsembleEDIST2 in handling different scenarios of complex drifts is weakly sensitive to N_0. Hence, we have decided to use $N_0 = 30$ as it has achieved the best accuracy rate in most cases.

Table 3. Prequential accuracy for different values of N_0 in EnsembleEDIST2

Complex drift	Gradual Continuous		Gradual Probabilistic		Abrupt	
	Local	Global	Local	Global	Local	Global
Synthetic database	Hyperplane	RBF	SEA Gradual	STAGGER Gradual	SEA Abrupt	STAGGER Abrupt
$N_0 = 30$	**98,6**	**95,9**	**97,2**	91,6	97,9	**99,6**
$N_0 = 60$	98,2	**95,9**	**97,2**	91,5	98,1	**99,6**
$N_0 = 90$	98,2	95,6	97,1	91,6	97,5	**99,6**
$N_0 = 120$	98,3	**95,9**	97,1	91,6	**98,2**	**99,6**
$N_0 = 150$	98,3	95,6	97,1	**91,7**	97,5	**99,6**

Impact of ensemble size on EnsembleEDIST2 performance EnsembleEDIST2 makes use of the parameter m in order to define the number of classifiers in the ensemble. Accordingly, it is interesting to study the impact of m on ensemble's performance according to different scenarios of complex drift.

According to Table4, it is noticeable that the size of EnsembleEDIST2 does not impact significantly the performance in handling different scenarios of complex drift. Hence, we have decided to use $m = 3$ as it achieved the best accuracy rate in most cases and it allows to limit the computational complexity of the ensemble.

Table 4. Accuracy of EnsembleEDIST2 with different number of base classifiers

Complex drift	Gradual Continuous		Gradual Probabilistic		Abrupt	
	Local	Global	Local	Global	Local	Global
Synthetic database	Hyperplane	RBF	SEA Gradual	STAGGER Gradual	SEA Abrupt	STAGGER Abrupt
$m = 3$	**98,6**	**95,9**	**97,2**	91,6	97,9	**99,6**
$m = 5$	**98,6**	**95,9**	97,1	91,6	**98**	**99,6**
$m = 10$	98.4	95,8	**97,2**	**91,1**	97,6	**99,6**

Accuracy of EnsembleEDIST2 Vs other ensembles Table5 summarizes the average of prequential accuracy during the drifting phase. The objective of this experiment is to study the ensemble performance in the presence of different scenarios of complex drift. Firstly, it is noticeable that EnsembleEDIST2 has achieved better results than *block-based ensembles* in handling different types of abrupt drift. During abrupt drift (independently of being local of global), the change is rapid; thus AUE, AWE and LearnNSE present difficulty in tuning off the block size to offer a compromise between fast reaction to drift and high accuracy. However, EnsembleEDIST2 is able to autonomously train ensemble members with variable amount of data at each time process, thus it can efficiently handle abrupt drift.

Secondly, it is noticeable that EnsembleEDIST2 outperforms *weighting-data ensembles* in handling different categories of global drift. During global drift (either continuous, probabilistic or abrupt), the change affects a large amount of data; thus when LeveragingBag and OzaBag intensify the re-use of data for training ensemble members, the performance's decrease is accentuated. In contrast, EnsembleEDIST2 duplicates only a set of filtered instances for training the ensemble members, that is why it is more accurate in handling global drift.

Thirdly, it is noticeable that EnsembleEDIST2 outperforms the *filtering-data ensembles* in handling different categories of local drift. During local drift (either continuous, probabilistic or abrupt), the change affects a little amount of data; thus the choice of the filtering criterion is a essential point for efficiently handling local drift. EnsembleEDIST2 defines a new filtering criterion, which is based on selecting the data that triggered the warning level. These data are the most representative of the new concept, thus when training the ensemble's members accordingly, it makes it more efficient for handling local drift.

EnsembleEDIST2 has also been tested through real world data sets which represent different scenarios of drift. It is worth underlining that the size of these data sets is relatively small comparing to the synthetic ones. Despite the different features of each real data set, encouraging results have been found where EnsembleEDIST2 has achieved the best accuracy in all the datasets (see Table6).

To sum, it is worth to underline that the combination of the three diversity techniques in EnsembleEDIST2 is beneficial for handling different scenarios of complex drift in the same time.

Table 5. Accuracy of EnsembleEDIST2 Vs. other ensembles in synthetic datasets

Complex Drift		Gradual Continuous		Gradual Probabilistic		Abrupt	
		Local	Global	Local	Global	Local	Global
Synthetic Dataset		Hyperplane	RBF	SEA Gradual	STAGGER Gradual	SEA Abrupt	STAGGER Abrupt
EnsembleEDIST2		**98,604**	**95,982**	**97,211**	**91,609**	**98,196**	**99,605**
Block-based	AUE	94,187	95,611	94,547	90,381	95,234	98,367
	AWE	94,054	95,018	94,563	90,551	95,23	98,367
	LearnNSE	96,369	95,44	94,372	85,873	95,079	39,049
Weighting-data	LeveragingBag	98,6	95,8	97,1	89,1	**98,2**	94,3
	OzaBag	98,195	93,533	96,982	69,21	98,132	96,64
Filtering-data	LimAttClass	91,281	94,186	91,126	86,553	91,226	94,893

Table 6. Accuracy of EnsembleEDIST2 Vs. other ensembles in real datasets

Real Dataset		Electricity	Spam
EnsembleEDIST2		**84,8**	**89,2**
Block-based	AUE	69,35	79,34
	AWE	72,09	60,25
	LearnNSE	72,07	60,33
Weighting-data	LeveragingBag	83.8	88,2
	OzaBag	82,3	82,7
Filtering-data	LimAttClass	82,6	63,9

6 Conclusion

In this paper, we have presented a new study of the role of diversity among the ensemble. More precisely, we have highlighted the advantages and the limits of three widely used diversity techniques (*block-based data*, *weighting-data* and *filtering data*) in handling *complex drift*.

Additionally, we have presented a new ensemble approach, namely EnsembleEDIST2, which combines these three diversity techniques. The intuition behind this approach is to explicitly handle drifts by using the drift detection mechanism EDIST2. Accordingly, the ensemble performance is monitored through a self-adaptive window. Hence, EnsembleEDIST2 can avoid the problem of tuning off the size of the batch data as it is the case of most block-based ensemble approaches, which is a desirable property for handling abrupt drifts. Secondly, it defines a new filtering criterion, which is based on selecting the data that trigger the warning level. Thanks to this property, EnsembleEDIST2 is more efficient for handling local drifts then conventional filtering-data ensembles, which are only based on filtering data according to similarity on feature space. Then, differently from the conventional weighting-data ensembles which apply the weighting process to all the data stream; EnsembleEDIST2 only intensifies the re-use of most representative data of the new concept, which is a desirable property for handling global drifts.

EnsembleEDIST2 has been tested different scenarios of complex drift. Encouraging results were found, comparing to similar approaches, where EnsembleEDIST2 has achieved the best accuracy rate in all datasets; and presented a stable behavior in handling different scenarios of complex drift.

It worth to underline that in the present investigation, the ensemble size, i.e., the number of ensemble members, was fixed. Hence it is interesting, for future work, to perform a strategy for dynamically adapting the ensemble size. The focus is that, during stable period, the ensemble size is maintained fixed; whereas during the drifting phase the size is autonomously adapted. This may ameliorate the performance and reduce the computational cost among the ensemble.

Acknowledgements The second author acknowledges the support of the Regional project REPAR, funded by the French Region Hauts-de-France.

7 EnsembleEDIST2 pseudo code

```
Algorithm EnsembleEDIST2
Input: $(x, y)$: Data Stream
        $N_0$: number of error to construct the window
        $m$: number of base classifier
Output: Trained ensemble classifier $E$
1.  for each base classifier $C_i$ from $E$
2.      $InitializeClassifier(C_i)$
3.  end for
4.  $W_G \leftarrow CollectInstances(E, N_0)$
5.  $W_{warning} \leftarrow \emptyset$
6.  repeat
7.      $W_0 \leftarrow CollectInstances(E, N_0)$
8.      $Level \leftarrow DetectedLevel(W_G, W_0)$
9.      switch (Level)
10.     case 1: $Incontrol$
11.         $W_G \leftarrow W_G \cup W_0$
12.         $UpdateParameters(W_G, W_0)$
13.         Increment all ensemble's members of $E$ according to instances in
            $W_G$
14.     end case 1
        case 2: $Warning$
15.         $W_{warning} \leftarrow W_{warning} \cup W_0$
16.         $UpdateParameters(W_{warning}, W_0)$
17.         $WeightingDataProcess(E, W_{warning})$
18.     end case 2
        case 3: $Drift$
19.         Create a new base classifier $C_{new}$ trained on instances in $W_{warning}$
20.         $E \leftarrow E \cup C_{new}$
21.         Remove the oldest classifier from $E$
22.         $W_G \leftarrow W_{warning}$
23.         $W_{warning} \leftarrow \emptyset$
24.     end case 3
25. end switch
26.until The end of the data streams
```

```
Algorithm DetectedLevel($W_G, W_0$)
Input: $W_G$: Global data window characterized by:
          $N_G$: error number
          $\mu_G$: error distance mean
          $\sigma_G$:error distance standard deviation
       $W_0$: Current data window characterized by:
          $N_0$: error number,
          $\mu_0$: error distance mean,
          $\sigma_0$:error distance standard deviation
**Output:** $Level$: detection level
1.  $\mu_d \leftarrow \mu_G - \mu_0$
2.  $\sigma_d \leftarrow \sqrt{\frac{\sigma_G^2}{N_G} + \frac{\sigma_0^2}{N_0}}$
3.  $\epsilon \leftarrow t_{1-\alpha} * \sigma_d$
4.  **if** $(\mu_d > \epsilon + \sigma_d)$
5.          $Level \leftarrow Drift$
6.          **else if** $(\mu_d > \epsilon)$
7.                  $Level \leftarrow Warning$
8.                  **else** $Level \leftarrow Incontrol$
9.          **end if**
10. **end if**
11. **return** (Level)
```

Algorithm *CollectInstances*(E, N_0)

```
**Input:** $(x, y)$: Data Stream
          $N_0$: number of error to construct the window
          $C$: trained ensemble classifier $E$
**Output:** $W$: Data window characterized by:
          $N$: error number
          $\mu$: error distance mean
          $\sigma$:error distance standard deviation
1.  $W \leftarrow \emptyset$
2.  $N \leftarrow 0$
3.  $\mu \leftarrow 0$
4.  $\sigma \leftarrow 0$
5.  **repeat** for each instance $x_i$
6.          $Prediction \leftarrow unweightedMajorityVote(E, x_i)$
7.          **if** $(Prediction = false)$
8.                  $d_i \leftarrow computeDistance()$
9.                  $\mu \leftarrow \frac{N}{N+1}\mu + \frac{d_i}{N+1}$
10.                 $\sigma \leftarrow \sqrt{\frac{N-1}{N}\sigma^2 + \frac{(d_i-\mu)^2}{N+1}}$
11.                 $N \leftarrow N + 1$
12.         **end if**
13.         $W \leftarrow W \cup \{x_i\}$
14. **until** $(N = N_0)$
15. **return** $(W)$
```

Algorithm *UpdateParameters*(W_G, W_0)

```
**Input:** $W_G$: Global data window characterized by:
          $N_G$: error number
```

```
        $\mu_G$: error distance mean
        $\sigma_G$:error distance standard deviation
    $W_0$: Current data window characterized by:
        $N_0$: error number,
        $\mu_0$: error distance mean,
        $\sigma_0$:error distance standard deviation
**Output:** Updated parameters of $W_G$
1.  $\mu_G \leftarrow \frac{1}{N_G+N_0}(N_G.\mu_G+N_0.\mu_0)$ $\sigma_G \leftarrow \sqrt{\frac{N_G\sigma_G^2+N_0\sigma_0^2}{N_G+N_0} + \frac{N_G N_0}{(N_G+N_0)^2}(\mu_G-\mu_0)^2}$
2.  $N_G \leftarrow N_G + N_0$
```

```
**Algorithm** *WeightingDataProcess($E, W_{warning}$)*
**Input:** $E$: Ensemble Classifier
        $W_{warning}$: Window of data
**Output:**     $E$: Updated ensemble classifier
1.  **for** each instance $x_i$ from $W_{warning}$
2.          **for** each base classifier $C_i$ from $E$
3.                  $k \leftarrow poisson(1)$
4.              **do** $k$ times
5.                  $TrainClassifier(C_i, x_i)$
6.                  **end do**
7.          **end for**
8.  **end for**
```

References

1. Bifet, A., Frank, E., Holmes, G., Pfahringer, B., Sugiyama, M., Yang, Q.: Accurate ensembles for data streams: Combining restricted hoeffding trees using stacking. In: 2nd Asian Conference on Machine Learning (ACML2010). pp. 225–240 (2010)
2. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. Journal of Machine Learning Research 11, 1601–1604 (2010)
3. Bifet, A., Holmes, G., Pfahringer, B.: Leveraging bagging for evolving data streams. In: Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part I. pp. 135–150. ECML PKDD'10, Springer-Verlag, Berlin, Heidelberg (2010), `http://dl.acm.org/citation.cfm?id=1888258.1888275`
4. Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavaldà, R.: New ensemble methods for evolving data streams. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 139–148. KDD '09, ACM, New York, NY, USA (2009), `http://doi.acm.org/10.1145/1557019.1557041`
5. Brzezinski, D., Stefanowski, J.: Reacting to different types of concept drift: The accuracy updated ensemble algorithm. Neural Networks and Learning Systems, IEEE Transactions on 25(1), 81–94 (Jan 2014)

6. Brzezinski, D., Stefanowski, J.: Accuracy updated ensemble for data streams with concept drift. In: Corchado, E., Kurzy?ski, M., Wo?niak, M. (eds.) Hybrid Artificial Intelligent Systems, Lecture Notes in Computer Science, vol. 6679, pp. 155–163. Springer Berlin Heidelberg (2011), `http://dx.doi.org/10.1007/978-3-642-21222-2_19`
7. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 71–80. KDD00, ACM, New York, NY, USA (2000)
8. Gama, J., Sebastio, R., Rodrigues, P.: On evaluating stream learning algorithms. Machine Learning 90(3), 317–346 (2013)
9. Harries, M.: Splice-2 comparative evaluation: Electricity pricing. Tech. rep., The University of South Wales, United Kingdom (1999)
10. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, CA, USA, August 26-29, 2001. pp. 97–106 (2001)
11. Katakis, I., Tsoumakas, G., Vlahavas, I.: Tracking recurring contexts using ensemble classifiers: an application to email filtering. Knowledge and Information Systems 22(3), 371–391 (2010)
12. Khamassi, I., Sayed-Mouchaweh, M.: Drift detection and monitoring in non-stationary environments. In: Evolving and Adaptive Intelligent Systems (EAIS), Austria. pp. 1–6 (June 2014)
13. Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., Ghédira, K.: Ensemble classifiers for drift detection and monitoring in dynamical environments. In: Annual Conference of the Prognostics and Health Management Society, New Orlean, USA, 2013 (October 2013)
14. Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., Ghédira, K.: Self-adaptive windowing approach for handling complex concept drift. Cognitive Computation 7(6), 772–790 (2015), `http://dx.doi.org/10.1007/s12559-015-9341-0`
15. Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., Ghédira, K.: Discussion and review on evolving data streams and concept drift adapting. Evolving Systems (Oct 2016), `http://dx.doi.org/10.1007/s12530-016-9168-2`
16. Kolter, J.Z., Maloof, M.A.: Dynamic weighted majority: An ensemble method for drifting concepts. J. Mach. Learn. Res. 8, 2755–2790 (Dec 2007)
17. Minku, L., White, A., Yao, X.: The impact of diversity on online ensemble learning in the presence of concept drift. Knowledge and Data Engineering, IEEE Transactions on 22(5), 730–742 (May 2010)
18. Minku, L., Yao, X.: Ddd: A new ensemble approach for dealing with concept drift. Knowledge and Data Engineering, IEEE Transactions on 24(4), 619–633 (April 2012)
19. Oza, N.C., Russell, S.: Online bagging and boosting. In: In Artificial Intelligence and Statistics 2001. pp. 105–112. Morgan Kaufmann (2001)
20. Polikar, R., Upda, L., Upda, S., Honavar, V.: Learn++: an incremental learning algorithm for supervised neural networks. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 31(4), 497–508 (Nov 2001)
21. Ren, Y., Zhang, L., Suganthan, P.N.: Ensemble classification and regression-recent developments, applications and future directions [review article]. IEEE Computational Intelligence Magazine 11(1), 41–53 (2016)
22. Sayed-Mouchaweh, M.: Learning from Data Streams in Dynamic Environments, chap. Handling Concept Drift, pp. 33–59. Springer International Publishing, Cham (2016), `http://dx.doi.org/10.1007/978-3-319-25667-2_3`

23. Schlimmer, J.C., Granger, Jr., R.H.: Incremental learning from noisy data. Mach. Learn. 1(3), 317–354 (Mar 1986)
24. Street, W.N., Kim, Y.: A streaming ensemble algorithm (sea) for large-scale classification. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 377–382. KDD01, ACM, New York, NY, USA (2001)

Comparison between Co-training and Self-training for single-target regression in data streams using AMRules

Ricardo Sousa[1] and João Gama[1,2]

[1] LIAAD/INESC TEC, Universidade do Porto, Portugal
rtsousa@inesctec.pt
[2] Faculdade de Economia, Universidade do Porto, Portugal
jgama@fep.up.pt

Abstract. A comparison between co-training and self-training method for single-target regression based on multiples learners is performed. Data streaming systems can create a significant amount of unlabeled data which is caused by label assignment impossibility, high cost of labeling or labeling long duration tasks. In supervised learning, this data is wasted. In order to take advantaged from unlabeled data, semi-supervised approaches such as Co-training and Self-training have been created to benefit from input information that is contained in unlabeled data. However, these approaches have been applied to classification and batch training scenarios.
Due to these facts, this paper presents a comparison between Co-training and Self-learning methods for single-target regression in data streams. Rules learning is used in this context since this methodology enables to explore the input information.
The experimental evaluation consisted of a comparison between the real standard scenario where all unlabeled data is rejected and scenarios where unlabeled data is used to improve the regression model.
Results show evidences of better performance in terms of error reduction and in high level of unlabeled examples in the stream. Despite this fact, the improvements are not expressive.

1 Introduction

Prediction represents an essential task in data streams contexts that depend on accurate predictions for decision making or planning [1]. In these contexts, large quantities of data is not labeled due to label assignment impossibility, high cost of label assignment or long time tasks. Frequently, sensitive data requires label omission [4].

The main areas where unlabeled data occurs are Engineering Systems (video object detection) [7], Physics (weather forecasting and ecological models) [8], Biology (model of cellular processes) [9] and Economy/Finance (stock price forecasting) [1]. In most of these areas, data from streams are obtained and processed in real time [4].

Semi-supervised Learning (SSL) methodology have been suggested to use input information from unlabeled data for more accurate predictions [4]. Only in unlabeled examples abundance cases, this methodology may be useful [11]. In fact, the unlabeled examples convey information related to the variety or to the range of the inputs values. These values ranges may create constrains to the models and them more precise. As negative characteristic, this methodology may introduce errors and lead to less accurate predictions [11, 12].

More formally, $\mathbf{X} = \{X_1, ..., X_j, ..., X_J\} \in \mathbb{R}^J$ represents a vector of input random variables and $\mathbf{Y}$ represents a scalar random variable, with a joint probability distribution $P(\mathbf{X}, \mathbf{Y})$. The vectors $\mathbf{x}_i = (x_{i,1}, ..., x_{i,j}, ..., x_{i,J}) \in \mathbb{R}^J$ and y_i, where $i \in \{0, 1, 2, ...\}$, represent realizations of $\mathbf{X}$ and $\mathbf{Y}$, respectively. A stream is defined as the sequence of examples $\mathbf{e}_i = (\mathbf{x}_i, y_i)$ represented as $\mathcal{S} = \{(\mathbf{x}_0, y_0), (\mathbf{x}_1, y_1), ..., (\mathbf{x}_i, y_i), ...\}$. Label absence is represented by $y_i = \emptyset$. The objective of SSL is to use examples $(\mathbf{x}_i, \emptyset)$ to enhance the regression model $y_i \leftarrow f(\mathbf{x}_i)$ and reduce the error of prediction for both labeled and unlabeled examples.

SSL methods work on batch mode and are applied to classification. The imediate adaption to regression is not possible [11]. Co-training is a SSL approach that uses more than one different models. The model diversity is created by through different inputs, different regression methods or different parametrization [4].

The training stage produces an artificial label for the unlabeled example from the regressors predictions of the same example, according to a criterion (e.g., mean of all predictions) [11]. Posteriorly, this artificially labeled example is used in the training of the regressors. The prediction stage yield a final prediction from the regressors predictions of the example, according to a similar criterion as in training.

Self-training can be seen as a particular case of Co-training where just one model is trained. This method uses its own predictions to artificially label the unlabeled examples and use this examples in the respective training.

It worths to say that the active learning can also be easily introduced in these methods. However, once used in the training, the example contribution cannot be removed from the model.

This work focus on the comparison between Self-training method and Co-training method which uses several models that learn with each other for online, single-target regression. Despite expecting better prediction results from Co-training, it is important to find how much the results are superior to Self-training. In fact, Co-training is more computationaly expensive then Self-training. This work also may pave the way for the extension to online multi-target regression using the Random Adaptive Model Rules (Random AMRules) algorithm in future works [15, 16].

This document is structured as follows. In Section 2, the fundamentals of SSL for Self-training and Co-training are briefly revised. Section 3 describes the modifications of the Co-training to online learning and regression using ensembles of rule models. Section 4 describes the evaluation method. The results are discussed in Section 5 and the main conclusions are remarked in Section 6.

2 Related Work

This section explains some concepts used in the development of Co-training and Self-training methods. As general pattern, Co-training involves the training of two or more different models in some aspects (e.g., different inputs, different regressors, different parametrization, different examples ...). The labeled examples are processed as a supervised processure. The unlabeled examples are artificially labeled and processed as a supervised procedure. The artificial label is essentially a prediction (Self-training) or a processed prediction derived from a combination of predictions of complementary learners (Co-training). The learners are considered to predict reliably (confidence driven method).

Co-training methods follow these assumptions: consensus, complementary, sufficiency, compatibility and conditional independence. Self-training only considers sufficiency and compatibility. Note that these assumptions can be applied for both batch and online(incremental) methods.

- **Consensus** assumption states that the more similar the learner predictions are, the more reliable the artificial label is [18].
- **Complementarity** assumption states the learners contain different information and can learn from each other [18].
- **Sufficiency** assumption states that each regressor should be sufficiently consistent (e.g., by enough number of attributes) to build a model.
- **Compatibility** assumption implies that the predictions of different models present the same probabilistic distribution.
- **Conditional independence** assumption gives the chance of at least one learner can produce a more accurate prediction.This prediction can be used to teach the other learners. [19].
- **Conditional independence** assumption considers that the learning process of each Despite being very important for Co-training, the independence assumption is very restrictive. Therefore, related but less restrictive assumptions were considered.
- **Weak dependence** assumption tolerates a small dependence level between inputs which lead to positive results. This assumption overcomes the restrictive characteristic of Conditional independence [20].
- **Large diversity** assumption considers that using different algorithms or the same algorithms but with different parametrization lead to independent models [21].

Concerning the drawbacks, the inaccuracy of the artificially labeled examples introduce error into the models and it is the main cause of model degradation. Moreover, the artificially labeled examples may not carry the information to the regressor leading to unnecessary operations [11]. Different strategies to artificially label or criteria to discard non-beneficial artificially labeled examples may be present in some Self-training and Co-training variants. The prediction stage generally combines the predictions of the models according to a pre-defined criterion to produce the final prediction [11].

3 Online Co-training and Self-training Regression

This section provides the description of a developed a Co-training method and also a Self-training method through the presentation of the main adaptations to the online and regression context. Here, the description was focused in the Co-training algorithm. A small description of the underlying algorithm regressor Random AMRules (ensemble rules based method) is also presented.

The new method that is being proposed divides the inputs variables of the example into two groups randomly which is defined in the initial stage. Here, weak dependence is assumed since no independence information between pairs of attributes is available. The complementarity assumption is also used since each produced model contains information that other does not contain.

The two groups are forced to share a randomly selected inputs by a pre-defined overlap percentage. Two Random AMRules complementary regressors are used to produce artificial labels through prediction for the unlabeled example. The initial models are obtained previously in a training stage using a dataset portion. The size of dataset portion should be sufficient to produce a consistent model. Here, the inputs overlapping increase the number of attributes in each model and contribute for the sufficiency assumption.

A score that reflects the benefit or confidence of artificially labeled example is calculated for the decision of being accepted for training. The score is the relative difference (RD) compared to the maximum of absolute values of the output found in the stream y_{max}. Here, the consensus assumption is used. Equation 1 defines de relative difference.

$$RD = \frac{|\hat{y}_i^1 - \hat{y}_i^2|}{y_{max}} \tag{1}$$

If the score is lower than a pre-defined threshold, the predictions are used to train the complementary regressor. Otherwise, the artificially labeled example is rejected. The consensus assumption is used in this step. If the example is labeled, this example is used to compute the mean error for each regressor. Next, the example is used for all regressors training. Here, the compatibility assumption is used since both models are trained with the same output. Algorithm 1 explains the training procedure of the proposed method.

Prediction is performed by combining the regressor predictions through prediction weighting. The weights are computed by inverting the values of the respective error produced by labeled examples in the training stage since the higher the error is, the less the artificial example benefits the model. In other words, this strategy gives more credit to the regressor that produces less errors. Algorithm 2 shows the steps of label prediction.

The Random AMRules regressor was employed to train the models and to produce the artificial labels for the unlabeled examples [16]. Random AMRules is a multi-target algorithm (predicts several outputs for the same example) that is based on rule learning which can be calibrated to work on single-target mode [3].

In essence, Random Rules is an ensemble based algorithm that uses bagging to create diversity and uses AMRules algorithm as a regressor. AMRules

Algorithm 1 Training algorithm of the proposed method

```
 1: Initialization:
 2:     $\alpha - Overlap\ percentage$
 3:     $s - Score\ Threshold$
 4:     Random input allocation and overlapping
 5:      into the two groups using $\alpha$
 6: Input: Example $(\mathbf{x}_i, y_i) \in \mathcal{S}$
 7: Output: Updated Models
 8: Method:
 9: Divide $\mathbf{x}_i$ into $\mathbf{x}_i^1$ and $\mathbf{x}_i^2$
10: if $(y_i = \emptyset)$ then
11:     $\hat{y}_i^1 = PredictModel1(\mathbf{x}_i^1)$
12:     $\hat{y}_i^2 = PredictModel2(\mathbf{x}_i^2)$
13:     if $(|\hat{y}_i^1 - \hat{y}_i^2|/y_{max} < s)$ then
14:         $TrainModel1((\mathbf{x}_i^1, \hat{y}_i^2))$
15:         $TrainModel2((\mathbf{x}_i^2, \hat{y}_i^1))$
16: else
17:     $\bar{e}_1 = Update\ the\ mean\ error\ of\ Model1(\hat{y}_i^1, y_i)$
18:     $\bar{e}_2 = Update\ the\ mean\ error\ of\ Model2(\hat{y}_i^2, y_i)$
19:     $TrainModel1((\mathbf{x}_i^1, y_i))$
20:     $TrainModel2((\mathbf{x}_i^2, y_i))$
```

Algorithm 2 Prediction algorithm of the proposed method

```
1: Input: Example $(\mathbf{x}_i, y_i) \in \mathcal{S}$
2: Output: Example prediction $\hat{y}_i$
3: Method:
4: Divide $\mathbf{x}_i$ into $\mathbf{x}_i^1$ and $\mathbf{x}_i^2$
5: $\hat{y}_i^1 = PredictModel1(\mathbf{x}_i^1)$
6: $\hat{y}_i^2 = PredictModel2(\mathbf{x}_i^2)$
7: $w_1 = \bar{e}_2/(\bar{e}_1 + \bar{e}_2))$
8: $w_2 = \bar{e}_1/(\bar{e}_1 + \bar{e}_2)$
9: $\hat{y}_i = w_1 * \hat{y}_i^1 + w_2 * \hat{y}_i^2$
```

divides the input space in order to train local model in each partition. AMRules partionates the input space and creates local models for each partition. The local models are trained using a single layer perceptron. Its main advantages are models simplicity, low computational cost and low error rates [3].

Modularity is one of the main advantages. In fact, this method allows the train of models for local input sections limited by the rule that are more precise. This algorithm also resorts to anomaly detection to avoid data outliers damage. Moreover, change detection on the stream is also employed by this method in order to avoid the influence of old information on the current predictions. The ensembles of AMRules can benefit the prediction by creating multiple and diverse regressor models by pruning the input partitions. The multiple regressor predictions create more possibilities to find a more accurate value. The ensembles also lead to a more stable final prediction and data change resilience.

The Self-training method basically consists of one learner that artificially labels the incoming example and uses directly in the training (without a rejection criteria). This method is very simple compared to Co-training, since only one model is trained and it doesnt present a condition for training acceptance. In terms of complexity, the Co-training presents the double complexity in required memory and computing power, when compared to the Self-Learning. In fact, the Co-training method basically trains two modesl while Self-training just trains one.

4 The Evaluation Method

The evaluation method and the material used in the experiments are described in this section. Real-world and artificial datasets were used to evaluate the proposed algorithm through a data stream simulation. A portion of 30% of the first examples of the stream were used for a initial consistent model training and the remaining 70% were used in the testing.

In order to produce an unlabeled examples in the test stage, a binary Bernoulli random process with a probability p was used to assign an example as labeled or unlabeled. In case of unlabeled assignment, the true output value is hidden from the algorithm. The p probabilities of unlabeled examples occurrence were 50%, 80%, 90%, 95% and 99%.

For the Co-training method, the score threshold values for algorithm calibration were 1×10^{-4}, 5×10^{-4}, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5 and 1. These values of score threshold are justified by the possibility of algorithm behaviour observation in multiple scales of this parameter. The overlap percentages assume the following values: 0%, 10%, 30%, 50%, 70% and 90%. The evaluation was performed in Prequential mode where in example arrival, the label prediction is performed first and then the example is used in the training [25]. Each Random AMRules regressor consists of ten regressors ensemble. This value was determined in a validation step were no significant improvement was observed above 10 regressors.

In these experiments, five real world and four artificial datasets were used to simulate the data stream. The real world datasets were House8L (Housing Data Set), House16L (Housing Data Set), CASP (Physicochemical Properties of Protein Tertiary Structure Data Set), California, blogDataTrain and the artificial datasets were 2dplanes, fried, elevators and ailerons. These datasets contain a single-target regression problem and are available at UCI repository [26].

Table 2 an shows the features of the real world and artificial data sets used in the method evaluation.

The performance measure used in these experiments was the mean relative error (MRE). The MRE is used as an intermediate measure to quantify the prediction precision of each test scenario for both labeled and unlabeled examples. The MRE Reduction (MRER) was measured by using the relative difference (in percentage) between the reference scenario (no unlabeled examples used) MRE_0 and the case with the parametrization that lead to the lowest error MRE_{lowest}

Table 1. Real world datasets description

Dataset	# Examples	# Inputs
House8L	22784	8
House16H	22784	16
calHousing	20640	7
CASP	45730	9
blogDataTrain	52472	281

Table 2. Artificial datasets description

Dataset	# Examples	# Inputs
2dplanes	40768	10
fried	40768	10
ailerons	13750	41
elevators	8752	18

(includes the reference case MRE_0). Equation 2 defines the MRER performance measure.

$$MRER = \frac{|MRE_0 - MRE_{lowest}|}{MRE_0}.100 \quad (\%) \tag{2}$$

If the reference case yields the lowest error, then the MRER is zero, which means that the algorithm is not useful for that particular scenario.

Massive Online Analysis (MOA) platform was used to accommodate the proposed algorithm [27]. This platform contains Machine Learning and Data Mining algorithms for data streams processing and was developed in JAVA programming language.

5 Results

In this section, the evaluation results for Co-training and Self-training are presented and discussed.

5.1 Co-training results

For each combination of overlap percentage and score threshold, the experiments were performed in 10 runs due to the fact that the inputs are selected randomly. This procedure is important to obtain more consistent values. The results also include the presentation of the MRER for each dataset and the unlabeled examples percentage simulation.

In the experiments was registered, for the particular case of overlap of 50% and score threshold of 0.001, the use of 9.1% of the unlabeled examples in the training lead to reduction of 3.85% of the MRE in average for the House16H dataset. In general, it was also observed that the overlapping decrease the MRE.

The failure in some scenario is explained by the fact of many unlabeled examples lead to model degradation and the artificial labels were very inaccurate (the curves of unlabeled examples scenarios are above the reference curve). This fact indicates that features of the datasets such as inputs variables distributions may dictate the performance.

This methods are prone to error propagation through the model. The error propagation through the model lead to worst predictions in the artificial labeling. This effect leads to a cycle that reinforce the error on each unlabeled example processing. In fact, the more unlabeled examples arrive the higher is the error.

Table 3 provides the MRER values of the experiments on real world datasets for each chosen unlabeled examples probabilities, for Co-training method.

Table 3. MRER (%) for real world datasets

Datasets	Unlabeled examples probabilities				
	50%	80%	90%	95%	99%
House8L	2,23	3,21	2,77	0,00	0,00
House16H	3,85	1,93	0,32	0,00	0,00
calHousing	2,37	2,02	0,75	0,01	0,00
CASP	0,80	1,65	0,00	0,00	0,00
blogDataTrain	1,17	0,40	0,37	0,00	0,00

Table 3 suggests that the proposed algorithm seems to improve the performance for most part of the scenarios. The Co-training method can produce error reduction in higher percentage of unlabeled examples than the Self-training method. Despite this fact, the MRER are in general relatively small but superior than the Self-training method.

Table 6 provides the MRER value for real artificial datasets in similar way as the real world datasets presented in Table 5.

Table 4. MRER (%) for artificial datasets

Datasets	Unlabeled examples probabilities				
	50%	80%	90%	95%	99%
2dplanes	2,39	0,90	0,75	0,00	0,00
fried	3,55	3,35	1,71	0,00	0,00
ailerons	2,67	1,79	0,01	0,95	0,00
elevators	1,35	1,11	0,71	0,00	0,00

The results on artificial datasets reinforce the same conclusions that were obtained from real world datasets. The MRER is similarly small.

The results show that for 99% of unlabeled examples probability, the method does not produce beneficial artificial labels. This high level of unlabeled examples in the stream represents an extreme scenario where the model is training almost with artificially labeled examples and the high error propagation can frequently occur.

5.2 Self-training results

Table 5 provides the MRER values of the experiments on real world datasets for each chosen unlabeled examples probabilities, for the Self-training method.

When MRER assumes the zero value, a combination of overlap percentage and score threshold values that improves the model was not found and the reference scenario presents the lower MRE.

Table 5. MRER (%) for real world datasets

Datasets	Unlabeled examples probabilities				
	50%	80%	90%	95%	99%
House8L	0,57	0,01	0,00	0,00	0,00
House16H	0,23	0,00	0,00	0,00	0,00
calHousing	0,02	0,00	0,00	0,00	0,00
CASP	0,00	0,00	0,00	0,00	0,00
blogDataTrain	0,44	0,00	0,00	0,00	0,00

Table 5 suggests that the proposed algorithm seems to improve the performance for few scenarios. In fact, the algorithm fails in high probabilities of unlabeled examples. Inclusively, there is one dataset that didnt produce any favourable result. In successful cases, the MRER are in general relatively small.

Table 6 provides the MRER value for real artificial datasets in similar way as the real world datasets presented in Table 5.

Table 6. MRER (%) for artificial datasets

Datasets	Unlabeled examples probabilities				
	50%	80%	90%	95%	99%
2dplanes	1,12	0,00	0,00	0,00	0,00
fried	0,17	0,00	0,00	0,00	0,00
ailerons	0,81	0,00	0,00	0,00	0,00
elevators	0,22	0,00	0,00	0,00	0,00

The results on artificial datasets also support the view that the more elevated the unlabeled probability is, the less is the benefit of the unlabeled examples. The MRER is similarly small and there are very few successfull cases.

These results show that Self-training is limited by the percentage of unlabeled in the stream. For unlabeled examples higher than 50 %, the Self-training does not produce any error reduction. This limitation is explained by the fact that the artificially labeled examples produce high errors which does not garantee compability of the predictions.

6 Conclusion

This paper addresses a comparison of an online Co-training and Self-training algorithm for single-target regression based on ensembles of rule models. This work is the base for the development of multi-target regression methodology capable of using unlabeled examples information for model improving.

The results support that Co-training approach which uses the Random AM-Rules method reduces the error with the appropriate parameters calibration. The main contribution was the overlapping and the consensus measure strategies that contribute to increase diversity and model consistency in a online co-training scenario. The comparison between Co-training and Self-training reveal that Co-training can in fact lead to higher error reductions that the Self-training. In addiction, Co-training can produce error reduction in higher level of unlabeled examples in the stream.

In fact, the MRER is positive when an amount of unlabeled examples are used in the training in most evaluation combinations. Despite this fact, the model benefit is still relatively small and the performance is highly dependent of a good parametrization tuning (score threshold and overlap percentage). In addition, the amount of unlabeled examples is relatively small to obtain some model improvement.

Considering future work, this work will be extended to multi-target regression. The fact that very few unlabeled examples can lead to some improvement may suggest the study of the conditions that lead to this improvement. To increase the method validity, future works will include a higher number of real world datasets with higher amount of examples. Datasets with particular features such drifts presence are also in view.

7 Acknowledgements

This work is financed under the project ”NORTE-01-0145-FEDER-000020” which was funded by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

References

1. Adebiyi A. Ariyo, Adewumi O. Adewumi, and Charles K. Ayo. Stock price prediction using the arima model. In *Proceedings of the 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, UKSIM '14, pages 106–112, Washington, DC, USA, 2014. IEEE Computer Society.
2. Changsheng Li, Weishan Dong, Qingshan Liu, and Xin Zhang. MORES: online incremental multiple-output regression for data streams. *CoRR*, abs/1412.5732, 2014.
3. João Duarte and João Gama. Multi-Target Regression from High-Speed Data Streams with Adaptive Model Rules. In *IEEE conference on Data Science and Advanced Analytics*, 2015.
4. Zhi hua Zhou, Senior Member, and Ming Li. Semi-supervised regression with co-training style algorithms. *IEEE Transactions on Knowledge and Data Engineering*, page 2007.
5. Nitesh V. Chawla and Grigoris Karakoulas. Learning from labeled and unlabeled data: An empirical study across techniques and domains. *J. Artif. Int. Res.*, 23(1):331–366, March 2005.
6. Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning.* The MIT Press, 1st edition, 2010.
7. Chuck Rosenberg, Martial Hebert, and Henry Schneiderman. Semi-supervised self-training of object detection models. In *Proceedings of the Seventh IEEE Workshops on Application of Computer Vision (WACV/MOTION'05) - Volume 1 - Volume 01*, WACV-MOTION '05, pages 29–36, Washington, DC, USA, 2005. IEEE Computer Society.
8. Zaid Chalabi Punam Mangtani Masahiro Hashizume Chisato Imai, Ben Armstrong. Article: Time series regression model for infectious disease and weather. *International Journal of Environment Research*, (142):319–327, June 2015.
9. Huseyin Sekerc Volkan Uslana. Article: Quantitative prediction of peptide binding afnity by using hybrid fuzzy support vector regression. *Applied Soft Computing*, (43):210–221, January 2016.
10. Jurica Levatić, Michelangelo Ceci, Dragi Kocev, and Sašo Džeroski. *Semi-supervised Learning for Multi-target Regression*, pages 3–18. Springer International Publishing, Cham, 2015.
11. Pilsung Kang, Dongil Kim, and Sungzoon Cho. Semi-supervised support vector regression based on self-training with label uncertainty: An application to virtual metrology in semiconductor manufacturing. *Expert Syst. Appl.*, 51:85–106, 2016.
12. Andrew B. Goldberg, Xiaojin Zhu, Alex Furger, and Jun-Ming Xu. Oasis: Online active semi-supervised learning. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pages 362–367. AAAI Press, 2011.
13. Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.
14. Amparo Albalate and Wolfgang Minker. *Semi-Supervised Classification Using Prior Word Clustering*, pages 91–125. John WileySons, Inc., 2013.
15. Ezilda Almeida, Petr Kosina, and João Gama. Random rules from data streams. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 813–814, New York, NY, USA, 2013. ACM.
16. João Duarte and João Gama. Ensembles of adaptive model rules from high-speed data streams. In *Proceedings of the 3rd International Conference on Big Data,*

Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications - Volume 36, BIGMINE'14, pages 198–213. JMLR.org, 2014.

17. Monica Bianchini, Marco Maggini, and Lakhmi C. Jain. *Handbook on Neural Information Processing*. Springer Publishing Company, Incorporated, 2013.
18. Chang Xu, Dacheng Tao, and Chao Xu. A survey on multi-view learning. *CoRR*, abs/1304.5634, 2013.
19. Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 92–100, New York, NY, USA, 1998. ACM.
20. Steven P. Abney. Bootstrapping. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA.*, pages 360–367, 2002.
21. Sally Goldman and Yan Zhou. Enhancing Supervised Learning with Unlabeled Data. In *Proc. 17th International Conf. on Machine Learning*, pages 327–334. Morgan Kaufmann, San Francisco, CA, 2000.
22. Mohamed Farouk Abdel Hady, Friedhelm Schwenker, and Günther Palm. Semi-supervised learning for regression with co-training by committee. In *Proceedings of the 19th International Conference on Artificial Neural Networks: Part I*, ICANN '09, pages 121–130, Berlin, Heidelberg, 2009. Springer-Verlag.
23. Ulf Brefeld, Thomas Gärtner, Tobias Scheffer, and Stefan Wrobel. Efficient co-regularised least squares regression. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 137–144, New York, NY, USA, 2006. ACM.
24. Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Trans. on Knowl. and Data Eng.*, 17(11):1529–1541, November 2005.
25. João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.
26. K. Bache and M. Lichman. UCI machine learning repository, 2013.
27. Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, August 2010.