Finding It Now: Networked Classifiers in Real-Time Stream Mining Systems

Raphael Ducasse, Cem Tekin, and Mihaela van der Schaar

Abstract The aim of this chapter is to describe and optimize the specifications of signal processing systems, aimed at extracting in real time valuable information out of large-scale decentralized datasets. A first section will explain the motivations and stakes and describe key characteristics and challenges of stream mining applications. We then formalize an analytical framework which will be used to describe and optimize distributed stream mining knowledge extraction from large scale streams. In stream mining applications, classifiers are organized into a connected topology mapped onto a distributed infrastructure. We will study linear chains and optimise the ordering of the classifiers to increase accuracy of classification and minimise delay. We then present a decentralized decision framework for joint topology construction and local classifier configuration. In many cases, accuracy of classifiers are not known beforehand. In the last section, we look at how to learn online the classifiers characteristics without increasing computation overhead. Stream mining is an active field of research, at the crossing of various disciplines, including multimedia signal processing, distributed systems, machine learning etc. As such, we will indicate several areas for future research and development.

R. Ducasse (✉)
The Boston Consulting Group, Boston, MA, USA
e-mail: ducasse.raphael@bcg.com

C. Tekin
Bilkent University, Ankara, Turkey
e-mail: cemtekin@ee.bilkent.edu.tr

M. van der Schaar
Oxford-Man Institute, Oxford, UK
University of California, Los Angeles, Los Angeles, CA, USA
e-mail: mihaela.vanderschaar@oxford-man.ox.ac.uk

S. S. Bhattacharyya et al. (eds.), *Handbook of Signal Processing Systems*,
https://doi.org/10.1007/978-3-319-91734-4_3

Fig. 1 Nine examples of high volume streaming applications

1 Defining Stream Mining

1.1 Motivation

The spread of computing, authoring and capturing devices along with high bandwidth connectivity has led to a proliferation of heterogeneous multimedia data including documents, emails, transactional data, digital audio, video and images, sensor measurements, medical data, etc. As a consequence, there is a large class of emerging stream mining applications for knowledge extraction, annotation and online search and retrieval which require operations such as classification, filtering, aggregation, and correlation over high-volume and heterogeneous data streams. As illustrated in Fig. 1, stream mining applications are used in multiple areas, such as financial analysis, spam and fraud detection, photo and video annotation, surveillance, medical services, search, etc.

Let us deep-dive into three illustrative applications to provide a more pragmatic approach to stream mining and identify key characteristics and challenges inherent to stream mining applications.

Fig. 2 Semantic concept detection in applications

1.1.1 Application 1: Semantic Concept Detection in Multimedia; Processing Heterogeneous and Dynamic Data in a Resource-Constrained Setting

Figure 2 illustrates how stream mining can be used to tag concepts on images or videos in order to perform a wide set of tasks, from search to ad-targeting. Based upon this stream mining framework, designers can construct, instrument, experiment with, and optimize applications that automatically categorize image and video data captured by various cameras into a list of semantic concepts (e.g., skating, tennis, etc.) using various chains of classifiers.

Importantly, such stream mining systems need to be highly adaptive to the dynamic and time-varying multimedia sequence characteristics, since the input stream is highly volatile. Furthermore, they must often be able to cope with limited system resources (e.g. CPU, memory, I/O bandwidth), working on devices such as smartphones with increasing power restrictions. Therefore, applications need to cope effectively with system overload due to large data volumes and limited system resources. Commonly used approaches to dealing with this problem in resource constrained stream mining are based on load-shedding, where algorithms determine when, where, what, and how much data to discard given the observed data characteristics, e.g. burst, desired Quality of Service (QoS) requirements, data value or delay constraints.

1.1.2 Application 2: Online Healthcare Monitoring; Processing Data in Real Time

Monitoring individual's health requires handling a large amount of data, coming from multiple sources such as biometric sensor data or contextual data sources. As shown in Fig. 3, processing this raw information, filtering and analyzing it are key challenges in medical services, as it allows real time census and detection of irregular condition. For example, monitoring pulse check enables to identify if patient is in critical condition.

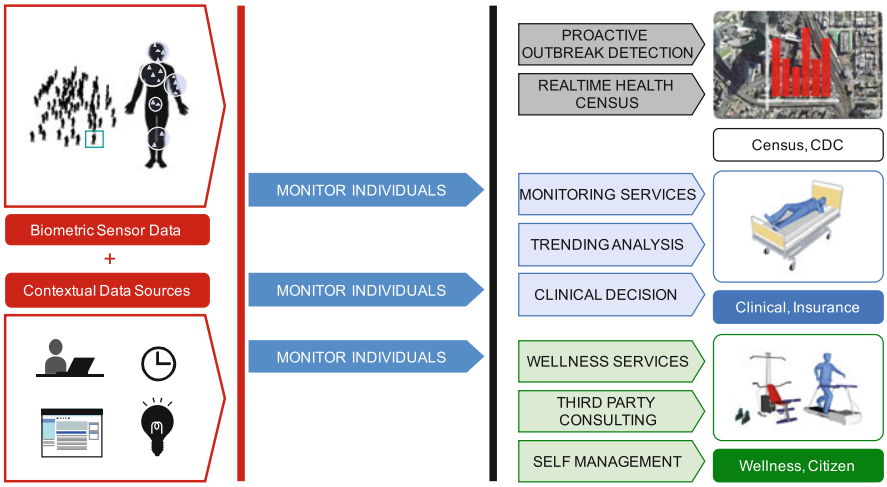

Fig. 3 Online healthcare monitoring workflow

In such application, being able to process data in real time is essential. Indeed, the information must be extracted and analyzed early enough to either take human decision or have an automatic control action. As an example, high concentration of calcium (happening under pain) could lead to either alerting medical staff or even automatic delivery of pain-killers, and the amount of calcium in the blood would determine the amount of medicine delivered. This control loop is only possible if the delay between health measurements (e.g. concentration of calcium in blood) and adaptation of treatment (e.g. concentration of pain-killer) is minimized.

1.1.3 Application 3: Analysis of Social Graphs; Coping with Decentralized Information and Setup

Social networks can be seen as a graph where nodes represent people (e.g. bloggers) and links represent interactions. Each node includes a temporal sequence of data, such as blog posts, tweets, etc. Numerous applications require to manage this huge amount of data: (1) selecting relevant content to answer keyword search, (2) identifying key influencers with page rank algorithms or SNA measures, and characterizing viral potential using followers' statistics, (3) recognizing objective vs. subjective content through lexical and pattern-based models, (4) automatically classifying data into topics (and creating new topics when needed) by observing work co-occurrence and using clustering techniques and classifying documents according to analysis performed on a small part of the document.

These applications are all the more challenging since the information is often decentralized across a very large set of computers, which is dynamically evolving over time. Implementing decentralized algorithms is therefore critical, even with

only partial information about other nodes. The performance of these algorithms can be greatly increased by using learning techniques, in order to progressively improve the pertinence of the analysis performed: at start, analysis is only based on limited data; over time, parameters of the stream mining application can be better estimated and the model used to process data is more and more precise.

1.2 From Data Mining to Stream Mining

1.2.1 Data Mining

Data mining can be described as the process of applying a ***query*** to a set of data, in order to select a sub-set of this data on which further action or analysis will be performed. For example, in Semantic Concept Detection, the query could be: "Select images of skating".

A data mining application may be viewed as a processing pipeline that analyzes data from a set of raw data sources to extract valuable information. The pipeline successively processes data through a set of filters, referred to as ***classifiers***. These classifiers can perform simple tests, and the query is the resultant of the answer of these multiple tests. For example, the query "Select images of skating" could be decomposed in the following tests: "Is it a team sport?"/"Is a Winter sport?"/"Is it a Ice sport?"/"Is it skating?"

Figure 4a provides an example of data mining application for sports image classification. Classifiers may be trained to detect different high-level semantic features, e.g. sports categories. In this example, the "Team Sports" classifier is used to filter the incoming data into two sets, thereby shedding a significant volume of data before passing it to the downstream classifiers (negatively identified team sports

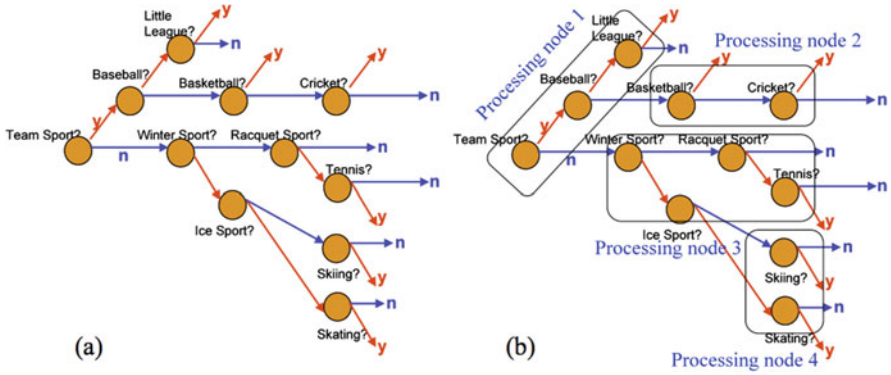

Fig. 4 A hierarchical classifier system that identifies several different sports categories and subcategories (**a**) at the same node, (**b**) across different nodes indicated in the figure as autonomous processing nodes

data is forwarded to the "Winter" classifier, while the remaining data is not further analyzed). Deploying a network of classifiers in this manner enables successive identification of multiple features in data, and provides significant advantages in terms of deployment costs. Indeed, decomposing complex jobs into a network of operators enhances scalability, reliability, and allows cost-performance tradeoffs to be performed. As a consequence, less computing resources are required because data is dynamically filtered through the classifier network. For instance, it has been shown that using classifiers operating in series with the same model (boosting [23]) or classifiers operating in parallel with multiple models (bagging [13]) can result in improved classification performance.

In this chapter, we will focus on mining applications that are built using a topology of low-complexity ***binary classifiers*** each mapped to a specific concept of interest. A binary classifier performs feature extraction and classification leading to a yes/no answer. However, this does not limit the generality of our solutions, as any M-ary classifiers may be decomposed into a chain of binary classifiers. Importantly, our focus will not be on the operators' or classifiers' design, for which many solutions already exist; instead, we will focus on configuring[1] the networks of distributed processing nodes, while trading off the processing accuracy against the available processing resources or the incurred processing delays. See Fig. 4b.

1.2.2 Changing Paradigm

Historically, mining applications were mostly used to find facts with data at rest. They relied on static databases and data warehouses, which were submitted to queries in order to extract and ***pull*** out valuable information out of raw data.

Recently, there has been a paradigm change in knowledge extraction: data is no longer considered static but rather as an inflowing stream, on which to dynamically compute queries and analysis in real time. For example, in Healthcare Monitoring, data (i.e., biometric measurements) is automatically analyzed through a batch of queries, such as "Verify that the calcium concentration is in the correct interval", "Verify that blood pressure is not too high", etc. Rather than applying a single query to data, the continuous stream of medical data is by default ***pushed*** through a predefined set of queries. This enables to detect any abnormal situation and react accordingly. See Fig. 5.

Interestingly, stream mining could lead to performing automatic action in response to a specific measurement. For example, a higher dose of pain killers could be administrated when concentration of calcium becomes too high, thus enabling real-time control. See Fig. 6.

[1] As we will discuss later, there are two types of configuration choices we must make: the topological ordering of classifiers and the local operating points at each classifier.

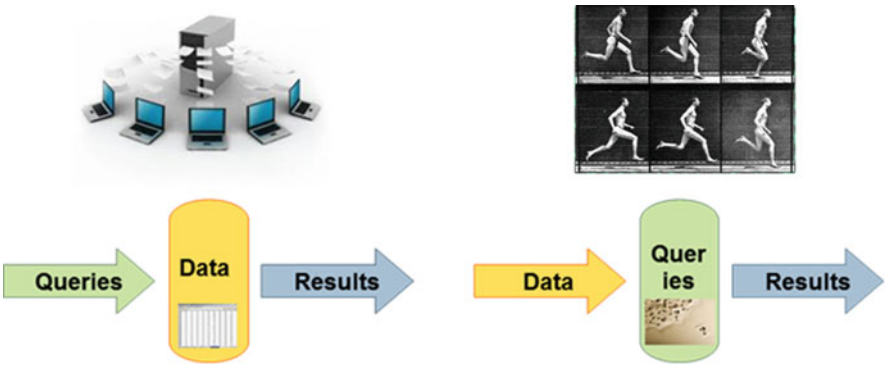

Fig. 5 A change of paradigm: continuous flow of information requires real-time extraction of insights

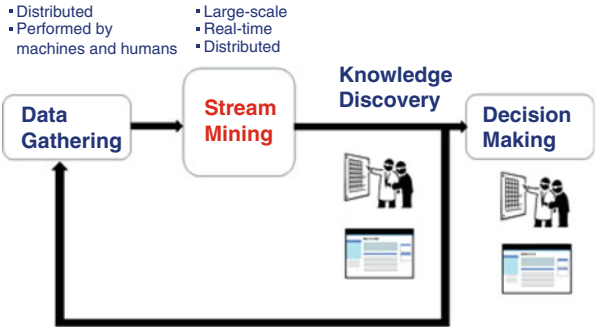

Fig. 6 Representation of knowledge extraction process in data mining system

1.3 Problem Formulation

1.3.1 Classifiers

A stream mining system can be seen as a set of binary classifiers. A binary classifier divides data into two subsets—one containing the object or information of interest (the "Positive" Set), and one not containing such objects or information (the "Negative" Set)—by applying a certain classification rule. For instance, the 'Team sport' classifiers separates images into those who represent a team sport and those who do not represent a team sport. This can be done using various classification techniques, such as Support Vector Machine (SVM), or K-nearest neighbor.

These algorithms are based on learning techniques, built upon test data and refined over time: they look for patterns in data, images, etc. and make decisions based on the resemblance of data to these patterns. As such, they are not fully accurate. A classifier can introduce two types of errors:

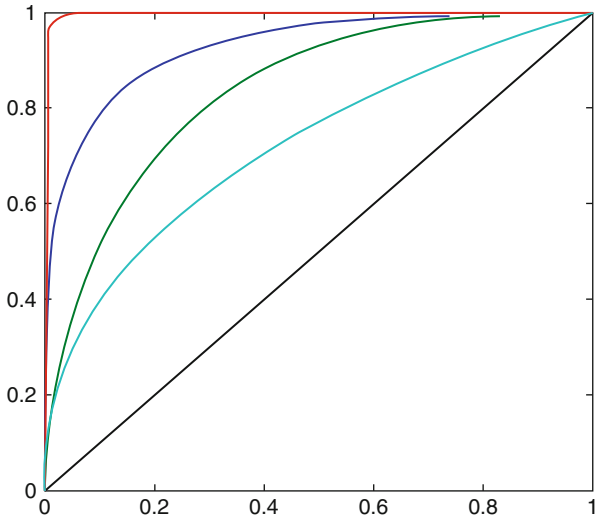

Fig. 7 ROC curves: $p^F = f(p^D)$. X axis is probability of misdetection error. Y axis is probably of false alarm error. We call sensitivity the factor that slides the operating point along the ROC curve

- Misdetection errors: Missing objects or data of interest by tagging it as belonging to the Negative Set rather than the Positive Set. We will note p^D the probability of detecting a data unit: $1 - p^D$ is the probability of misdetection.
- False alarm errors: Wrongly tagging objects or data which are not of interest as belonging to the Positive Set. We will note p^F this probability of false alarm.

Naturally, there is a trade-off between misdetection and false alarm errors: to avoid misdetections, the classifier could tag all data as positive, which would generate a high false alarm rate.

We will call **operating point** the couple (p^D, p^F). In Fig. 7, the operating points of various classifiers are plotted and form what is referred as ROC curves. The accuracy of the classifier depends on the concavity of the ROC curve, the more concave, the more precise.

The operating points' choice has two consequences on the performance of the stream mining system. First, it affects the precision of each classifier (both misdetection and false alarms) and of the system as a whole. Secondly, it defines the amount of data which is going to be transmitted through the classifiers and therefore the delay required for the system to process the data stream.

1.3.2 Axis for Study

This chapter focuses on developing a new systematic framework for knowledge extraction from high-volume data streams using a network of classifiers deployed

over a distributed computing infrastructure. It can be decomposed into four sub-problems which we will develop in the following sections:

1. **Stream Mining System Optimization:** In Sect. 2, we develop optimization techniques for tuning the operating points of individual classifiers in order to improve the stream mining performance, in terms of accuracy and delay. We formalize the problem of large-scale knowledge extraction by defining appropriate local and end-to-end objective functions, along with resource and delay constraints. They will guide the optimization and adaptation algorithms used to improve the stream mining performance.
2. **Stream Mining System Topology Optimization:** As shown in Fig. 4, a stream mining system is a topology of classifiers mapped onto a distributed infrastructure. These classifiers can be organized in one single chain, or in multiple parallel chains, thus forming a tree topology. In Sect. 3, we investigate the impact of the classifiers' topology on the performance, scalability and dynamic behavior of the stream mining system. We will focus on the study of linear chains of classifiers and determine how to jointly choose the order of classifiers in the chain and the operating point of each classifier in order to maximize accuracy and minimize delays.
3. **Decentralized Solutions Based on Interactive Multi-Agent Learning:** For large scale stream mining systems, where the classifiers are distributed across multiple nodes, the choice of operating point and topology of the classifiers would require heavy computational resources. Furthermore, optimizing the overall performance requires interactive multi-agent solutions to be deployed at each node in order to determine the effect of each classifiers' decisions on the other classifiers and hence, the end to end performance of the stream mining applications. In the fourth section of this chapter, we develop a decentralized decision framework for stream mining configuration and propose distributed algorithms for joint topology construction and local classifier configuration. This approach will cope with dynamically changing environments and data characteristics and adapt to the timing requirements and deadlines imposed by other nodes or applications.
4. **Online Learning for Real-Time Stream Mining:** In Sect. 5, we consider the stream mining problems in which the classifier accuracies are not known beforehand and needs to be learned online. Such cases frequently appear in real applications due to the dynamic behavior of heterogeneous data streams. We explain how the best classifiers (or classifier configurations) can be learned via repeated interaction, by driving the classifier selection process using meta-data. We also model the loss due to not knowing the classifier accuracies beforehand using the notion of regret, and explain how the regret can be minimized while ensuring that memory and computation overheads are kept at reasonable levels.

1.4 Challenges

Several key research challenges drive our analysis and need to be tackled: These are discussed in the following sections.

1.4.1 Coping with Complex Data: Large-Scale, Heterogeneous and Time-Varying

First, streaming data supposes that have high volume of timeless information flows in continuously. Stream mining systems thus need be scalable to massive data source and be able to simultaneously deal with multiple queries.

Both structured and unstructured data may be mined. In practice, data is wildly heterogeneous in terms of formats (documents, emails, transactions, digital video and/or audio data, RSS feeds) as well as data rates (manufacturing: 5–10 Mbps, astronomy: 1–5 Gbps, healthcare: 10–50 Kbps per patient). Furthermore, data sources and sensors may eventually be distributed on multiple processing nodes, with little or no communication in between them.

Stream mining systems need to be adaptive in order to cope with data and configuration dynamics: (1) heterogeneous data stream characteristics, (2) classifier dependencies, (3) congestion at shared processing nodes and (4) communication delays between processing nodes. Additionally, several different queries (requiring different topological combinations of classifiers) may need to be satisfied by the system, requiring reconfiguration as queries change dynamically.

1.4.2 Immediacy

Stream mining happens **now**, in real time. The shift from data mining to stream mining supposes that data cannot be stored and has to be processed on the fly.

For instance, in healthcare monitoring, minimizing delay between health measurements (e.g. concentration of calcium in blood) and adaptation of treatment (e.g. concentration of pain-killer) is critical. For some applications such as high-frequency trading, being real time may even be more important than minimizing misclassification costs. otherwise historic data would become obsolete and lead to phrased-out investment decisions.

Delay has seldom been analyzed in existing work on stream mining systems and, when it has been [1], it has always been analyzed in steady-state, at equilibrium, after all processing nodes are configured. However, the equilibrium can often not be reached due to the dynamic arrival and departure of query applications. Hence, this reconfiguration delay out of equilibrium must be considered when designing solutions for real-time stream mining systems.

Delay constraints are all the more challenging in a distributed environment, where the synchronization among nodes may not be possible or may lead to sub-optimal designs, as various nodes may experience different environmental dynamics and demands.

1.4.3 Distributed Information and Knowledge Extraction

To date, a majority of approaches for constructing and adapting stream mining applications are based on centralized algorithms, which require information about each classifier's analytics to be available at one node, and for that node to manage the entire classifier network. This limits scalability, creates a single point of failure, and provide limits in terms of adaptivity to dynamics.

Yet, data sources and classifiers are often distributed over a set of processing nodes and each node of the network may exchange only limited and/or costly message with other interconnected nodes to. Thus, it may be impractical to develop centralized solutions [4, 7, 18, 32, 33].

In order to address this naturally distributed setting, as well as the high computational complexity of the analytics, it is required to formally define local objectives and metrics and to associate inter-node message exchanges that enable the decomposition of the application into a set of autonomously operating nodes, while ensuring global performance. Such ***distributed mining systems*** have recently been developed [5, 19]. However, they do not encompass the accuracy and delay objectives described earlier.

Depending on the system considered, classifiers can have strong to very limited communication. Thus, classifiers may not have sufficient information to jointly configure their operating points. In such distributed scenarios, optimizing the end-to-end performance requires interactive, multi-agent solutions in order to determine the effect of each classifier's decisions on the other classifiers. Nodes need to learn online the effect of both their experienced dynamics as well as the coupling between classifiers.

Besides, for classifiers instantiated on separate nodes (possibly over a network), the communication time between nodes can greatly increase the total time required to deal with a data stream. Hence, the nodes will not be able to make decisions synchronously.

1.4.4 Resource Constraints

A key research challenge [1, 12] in distributed stream mining systems arises from the need to cope effectively with system overload, due to limited system resources (e.g. CPU, memory, I/O bandwidth etc.) while providing desired application performance. Specifically, there is a large computational cost incurred by each classifier (proportional to the data rate) that limits the rate at which the application can handle input data. This is all the more topical in a technological environment where low-power devices such as smartphones are becoming more and more used.

2 Proposed Systematic Framework for Stream Mining Systems

2.1 Query Process Modeled as Classifier Chain

Stream data analysis applications pose queries on data that require multiple concepts to be identified. More specifically, a query q is answered as a conjunction of a set of N classifiers $\mathcal{C}(q) = \{C_1, \ldots, C_N\}$, each associated with a concept to be identified (e.g. Fig. 4 shows a stream mining system where the concepts to be identified are sports categories).

In this chapter, we focus on binary classifiers: each binary classifier C_i labels input data into two classes $\mathcal{H}_i$ (considered without loss of generality as the class of interest) and $\overline{\mathcal{H}_i}$. The objective is to extract data belonging to $\bigcap_{i=1}^{N} \mathcal{H}_i$.

Partitioning the problem into this ensemble of classifiers and filtering data successively (i.e. discarding data that is not labelled as belonging to the class of interest), enables to control the amount of resources consumed by each classifier in the ensemble. Indeed, only data labelled as belonging to $\mathcal{H}_i$ is forwarded, while data labelled as belonging to $\overline{\mathcal{H}_i}$ is dropped. Hence, a classifier only has to process a subset of the data processed by the previous classifier. This justifies using a chain topology of classifiers, where the output of one classifier C_{i-1} feeds the input of classifier C_i, and so on, as shown in Fig. 8.

2.1.1 A-Priori Selectivity

Let X represent the input data of a classifier C. We call *a-priori* selectivity $\phi = \mathbf{P}(X \in \mathcal{H})$ the *a-priori* probability that the data X belongs to the class of interest. Correspondingly $1 - \phi = \mathbf{P}(X \in \overline{\mathcal{H}})$. Practically speaking, the *a-priori* selectivity ϕ is computed on a training and cross-validation data set. For well-trained classifiers, it is reasonable to expect that the performance on new, unseen test data is similar to that characterized on training data. In practice, there is potential train-test mismatch in behavior, but this can be accounted for using periodic reevaluation of the classifier performance (e.g. feedback on generated results).

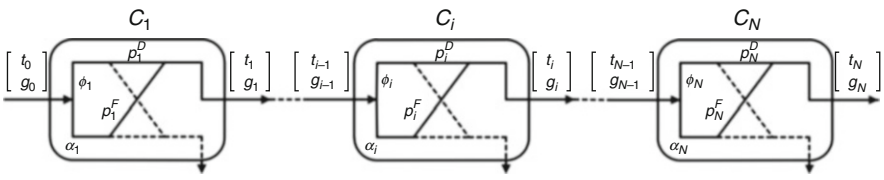

Fig. 8 Representation of analytical framework to evaluate classifier chain performance

For a chain of classifiers $\mathcal{C} = \{C_1, \ldots, C_N\}$, the *a-priori* selectivity of a classifier corresponds to the conditional probability of data belonging to classifier C_i's class of interest, given that it belongs to the class of interest of the previous $i-1$ classifiers: $\phi_i = \mathbf{P}(X \in \mathcal{H}_i | X \in \bigcap_{k=1}^{i-1} \mathcal{H}_k)$. Similarly, we define the negative *a-priori* selectivity as $\overline{\phi_i} = \mathbf{P}(X \in \mathcal{H}_i | X \notin \bigcap_{k=1}^{i-1} \mathcal{H}_k)$. Since *a-priori* selectivities depend on classifiers higher in the chain, $\overline{\phi_i} \neq 1 - \phi_i$.

2.1.2 Classifier Performance

The output $\hat{X}$ of a classifier C can be modeled as a probabilistic function of its input X. The proportion of correctly classified samples in $\mathcal{H}_k$ is captured by the probability of correct detection $p_k^D = \mathbf{P}(\hat{X} \in \mathcal{H}_k | X \in \mathcal{H}_k)$, while the proportion of falsely classified samples in $\mathcal{H}_k$ is $p_k^F = \mathbf{P}(\hat{X} \in \mathcal{H}_k | X \in \overline{\mathcal{H}_k})$.

The performance of the classifier C is characterized by its ROC curve that represents the tradeoff between the probability of detection p^D and probability of false alarm p^F. We represent the ROC curve as a function $f : p^F \mapsto p^D$ that is increasing, concave and lies over the first bisector [11]. As a consequence, an operating point on this curve is parameterized uniquely by its false alarm rate $x = p^F$. The operating point is denoted by $(x, f(x)) = (p^F, p^D)$.

We model the average time needed for classifier C to process a stream tuple as α (in seconds). The order of magnitude of α depends on the data characteristics, as well as the classification algorithm, and can vary from microseconds (screening text) to multiple seconds (complex image or video classification).

2.1.3 Throughput and Goodput of a Chain of Classifiers

The forwarded output of a classifier C_i consists of both correctly labelled data from class $\mathcal{H}_i$ as well as false alarms from class $\overline{\mathcal{H}_i}$. We use g_i to represent the goodput (portion of data correctly labelled) and t_i to represent the throughput (total forwarded data, including mistakes). And we will note t_0 to represent the input rate of data.

Using Bayes formula, we can derive t_i and g_i recursively as

$$\begin{bmatrix} t_i \\ g_i \end{bmatrix} = \underbrace{\begin{bmatrix} a_i & b_i \\ 0 & c_i \end{bmatrix}}_{T_i^{i-1}} \begin{bmatrix} t_{i-1} \\ g_{i-1} \end{bmatrix}, \qquad \text{where} \qquad \begin{cases} a_i = p_i^F + (p_i^D - p_i^F)\overline{\phi_i} \\ b_i = (p_i^D - p_i^F)(\phi_i - \overline{\phi_i}) \\ c_i = p_i^D \phi_i \end{cases} \tag{1}$$

For a set of independent classifiers, the positive and negative *a-priori* selectivities are equal: $\phi_i = \overline{\phi_i} = \mathbf{P}(X \in \mathcal{H})$. As a consequence, the transition matrix is diagonal: $T_i^{i-1} = \begin{bmatrix} p_i^D \phi_i + (1-\phi_i) p_i^F & 0 \\ 0 & p_i^D \phi_i \end{bmatrix}$.

2.2 Optimization Objective

The global utility function of the stream mining system can be expressed as a function of misclassification and delay cost, under resource constraints.

2.2.1 Misclassification Cost

The misclassification cost, or error cost, may be computed in terms of the two types of accuracy errors—a penalty c^M per unit rate of missed detection, and a penalty c^F per unit rate of false alarm. These are specified by the application requirements. Noting $\Phi = \prod_{h=1}^{N} \phi_h$, the total misclassification cost is

$$c_{err} = c^M \underbrace{(\Phi t_0 - g_N)}_{misseddata} + c^F \underbrace{(t_N - g_N)}_{wronglyclassifieddata} . \tag{2}$$

2.2.2 Processing Delay Cost

Delay may be defined as the time required by the chain of classifiers in order to process a stream tuple. Let α_i denote the expected processing time of classifier C_i. The average time required by classifier C_i to process a stream tuple is given by $\delta_i = \alpha_i P_i$, where P_i denotes the fraction of data which has not been rejected by the first $i-1$ classifiers and still needs to be processed through the remaining classifiers of the chain. Recursively, $P_i = \prod_{k=1}^{i-1} \frac{t_k}{t_{k-1}} = \frac{t_{i-1}}{t_0}$. After summation across all classifiers, the average end-to-end processing time required by the chain to process stream data is

$$c_{delay} = t_0 \sum_{i=1}^{N} \delta_i = t_0 \sum_{i=1}^{N} \alpha_i P_i = \sum_{i=1}^{N} \alpha_i t_{i-1}. \tag{3}$$

2.2.3 Resource Constraints

Assume that the N classifiers are instantiated on M processing nodes, each of which has a given available resource r_j^{max}. We can define a location matrix $M \in \{0, 1\}^{M \times N}$ where $M_{ji} = 1$ if C_i is located on node j and 0 otherwise. The resource constraint at node j can be written as $\sum_{i=1}^{N} M_{ji} r_i \leq r_j^{max}$.

The resource r_i consumed at node j by classifier C_i is proportional to the throughput t_i, i.e. $r_i \propto t_i$.

2.2.4 Optimization Problem

Stream mining system configuration involves optimizing both accuracy and delay under resource constraints. The utility function of this optimization problem may be defined as the negative weighted sum of both the misclassification cost and the processing delay cost: $U = -c_{err} - \lambda\, c_{delay}$, where the parameter λ controls the tradeoff between misclassification and delay. This utility is a function of the throughputs and goodputs of the stream within the chain, and therefore implicitly depends on the operating point $x_i = p_i^F \in [0, 1]$ selected by each classifier.

Let $\mathbf{x} = \left[x_1, \ldots, x_N\right]^T$, $K = \frac{c^F}{c^F + c^M} \in [0, 1]$ and $\rho = \frac{\lambda}{c^F + c^M}\boldsymbol{\alpha} \in \mathbf{R}^{+N}$. The optimization problem can be reformulated under a canonic format as follows:

$$\begin{cases} \underset{\mathbf{x}\in[0\ 1]^N}{\text{maximize}}\ U(\mathbf{x}) = g_N(\mathbf{x}) - K t_N(\mathbf{x}) - \sum_{i=1}^{N} \rho_i t_{i-1}(\mathbf{x}) \\ \text{subject to } \mathbf{0} \le \mathbf{x} \le \mathbf{1} \quad \text{and } M\mathbf{r} \le \mathbf{r}^{\max} \end{cases}. \tag{4}$$

2.3 Operating Point Selection

Given a topology, the resource-constrained optimization problem defined in Eq. (4) may be formulated as a network optimization problem (NOP) [16, 20]. This problem has been well studied in [11, 21, 31] and we refer the interested reader to the corresponding literature.

The solutions proposed involve using iterative optimization techniques based on Sequential Quadratic Programming (SQP) [3]. SQP is based on gradient-descent, and models a nonlinear optimization problem as an approximate quadratic programming subproblem at each iteration, ultimately converging to a locally optimal solution.

Selecting the operating point can be done by applying the SQP-algorithm to the Lagrangian function of the optimization problem in (4):

$$\mathcal{L}(\mathbf{x}, \nu_1, \nu_2) = U(\mathbf{x}) - \nu_1^T(\mathbf{x} - 1) + \nu_2^T\mathbf{x}.$$

Because of the gradient-descent nature of the SQP algorithm, it is not possible to guarantee convergence to the global maximum and the convergence may only be locally optimal. However, the SQP algorithm can be initialized with multiple starting configurations in order to find a better local optimum (or even the global optimum). Since the number and size of local optima depend on the shape of the various ROC curves of each classifier, a rigorous bound on the probability to find the global optimum cannot be proven. However, certain start regions are more likely to converge to better local optimum.[2]

[2]For example, since the operating point $p^F = 0$ corresponds to a saddle point of the utility function, it would achieve steepest utility slope. Furthermore, the slope of the ROC curve is

2.4 Further Research Areas

Further research areas are the following:

- **Communication delay between classifiers:** The model could be further refined to explicitly consider communication delays, i.e. the time needed to send stream tuples from one classifier to another. This is all the more true in low-delay settings where classifiers are instantiated on different nodes.
- **Queuing delay between classifiers:** Due to resource constraints, some classifiers may get congested, and the stream will hence incur additional delay. Modeling these queuing delays would further improve the suitability of the framework for real-time applications.
- **Single versus multiple operating points per classifier:** Performance gains can be achieved by allowing classifiers to have different operating points x_i and $\overline{x_i}$ for their positive and negative classes. If the two thresholds overlap, low-confidence data will be duplicated across both output edges, thereby increasing the end-to-end detection probability. If they do not overlap, low-confidence data is shed, thus reducing congestion at downstream classifiers.
- **Multi-query optimization:** Finally, a major research area would consist in studying how the proposed optimization and configuration strategies adapt to multi-query settings, including mechanisms for admission control of queries.

3 Topology Construction

In the previous section, we have determined how to improve performance of a stream mining system—both in terms of accuracy and delays—by selecting the right operating point for each classifier of the chain. This optimization was however performed given a specific topology of classifiers: classifiers were supposed arranged as a chain and the order of the classifiers in the chain was fixed.

In this section, we study the impact of the topology of classifiers on the performance of the stream mining system. We start by focusing on a chain topology and study how the order of classifiers on the chain alters performance.

3.1 Linear Topology Optimization: Problem Formulation

Since classifiers have different *a-priori* selectivities, operating points, and complexities, different topologies of classifiers will lead to different classification and delay costs.

maximal at $p^F = 0$ (due to concavity of the ROC curve), such that high detection probabilities can be obtained under low false alarm probabilities near the origin.

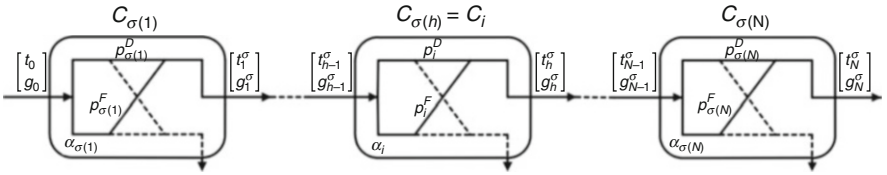

Fig. 9 Representation of σ-ordered classifier chain

Consider N classifiers in a chain, defined as in the previous section. An order $\sigma \in \mathcal{P}erm(N)$ is a permutation such that input data flows from $C_{\sigma(1)}$ to $C_{\sigma(N)}$. We generically use the index i to identify a classifier and h to refer to its depth in the chain of classifiers. Hence, $C_i = C_{\sigma(h)}$ will mean that the hth classifier in the chain is C_i. To illustrate the different notations used, a σ-ordered classifier chain is shown in Fig. 9.

Using the recursive relationship defined in Eq. (1), we can derive the end-to-end throughput t_i and goodput g_i of classifier $C_i = C_{\sigma(h)}$ recursively as

$$\begin{bmatrix} t_i \\ g_i \end{bmatrix} = \underbrace{\begin{bmatrix} p_i^F + \overline{\phi_h^\sigma}(p_i^D - p_i^F) & (\phi_h^\sigma - \overline{\phi_h^\sigma})(p_i^D - p_i^F) \\ 0 & \phi_h^\sigma p_i^D \end{bmatrix}}_{T_i^{i-1} = T_h^\sigma} \begin{bmatrix} t_{h-1}^\sigma \\ g_{h-1}^\sigma \end{bmatrix}. \tag{5}$$

The optimization problem can be written as:

$$\begin{cases} \underset{\sigma \in \mathcal{P}erm(N), \mathbf{x} \in [0\ 1]^N}{\text{maximize}} & U(\sigma, \mathbf{x}) = g_N^\sigma(\mathbf{x}) - K t_N^\sigma(\mathbf{x}) - \sum_{i=1}^{N} \rho_i t_{i-1}^\sigma(\mathbf{x}) \\ \text{subject to} & \mathbf{0} \le \mathbf{x} \le \mathbf{1} \end{cases}. \tag{6}$$

3.2 Centralized Ordering Algorithms for Fixed Operating Points

In this section, we consider a set of classifiers with fixed operating points $\mathbf{x}$. Since transition matrices T_i^σ are lower triangular, the goodput does not depend on the order of classifiers.[3] As a consequence, the expression of the utility defined in Eq. (4) can be simplified as:

[3]Furthermore, when classifiers are independent, the transition matrices T_i^σ are diagonal and therefore commute. As a consequence the end throughput $t_N(\mathbf{x})$ and goodput $g_N(\mathbf{x})$ are independent of the order. However, intermediate throughputs do depend on the ordering—leading to varying expected delays for the overall processing.

$$\underset{\sigma \in \mathcal{P}erm([1,N])}{\text{maximize}} \; U_{ord} = -\left(\sum_{h=1}^{N} \rho_{\sigma(h)} t_{h-1}^{\sigma} + K t_N^{\sigma} \right). \tag{7}$$

3.2.1 Optimal Order Search

The topology construction problem involves optimizing the defined utility by selecting the appropriate order σ. In general, there exist $N!$ different topologic orders, each with a different achieved utility and processing delay. Furthermore, the relationship between order and utility cannot be captured using monotonic or convex analytical functions. Hence, any search space for order selection increases combinatorially with N. This problem is exacerbated in dynamic settings where the optimal order has to be updated online; in settings with multiple chains, where each chain has to be matched with a specific optimal order; and, in settings with multiple data streams corresponding to the queries of multiple users.

3.2.2 Greedy Algorithm

Instead of solving the complex combinatorial problem, we suggest to design simple, but elegant and powerful, order selection algorithms—or Greedy Algorithms—with provable bounds on performance [2, 6].

The Greedy Algorithm is based on the notion of *ex-post* selectivity. For a given order σ, we define the *ex-post* selectivity as the conditional probability of classifier $C_{\sigma(h)}$ labelling a data item as positive given that the previous $h-1$ classifiers labelled the data as positive,[4] i.e. $\psi_h^{\sigma} = \frac{t_h^{\sigma}}{t_{h-1}^{\sigma}}$. The throughput at each step can be expressed recursively as a product of *ex-post* selectivities: $t_h^{\sigma} = \psi_h^{\sigma} t_{h-1}^{\sigma} = \ldots = \left(\prod_{i=1}^{h} \psi_i^{\sigma} \right) t_0$.

The Greedy Algorithm then involves ordering classifiers in increasing order of $\frac{\psi}{\mu}$ where $\mu_i^{\sigma} = \begin{cases} \rho_{\sigma(i+1)} = \frac{\lambda}{c^M + c^F} \alpha_{\sigma(i+1)} & \text{if } i \leq N-1 \\ K = \frac{c^F}{c^F + c^M} & \text{if } i = N \end{cases}$. Note that this fraction depends on the selected order.

Since this ratio depends implicitly on the order of classifiers in the chain, the algorithm may be implemented iteratively, selecting the first classifier, then selecting the second classifier given the fixed first classifier, and so on:

[4] Observe that for a perfect classifier ($p_{\sigma(h)}^{D} = 1$ and $p_{\sigma(h)}^{F} = 0$), the *a-priori* conditional probability ϕ_h^{σ} and the *ex-post* conditional probabilities ψ_h^{σ} are equal.

Centralized Algorithm 1 Greedy ordering

- Calculate the ratio $\psi_1^\sigma/\mu_1^\sigma$ for all N classifiers. Select $C_{\sigma(1)}$ as the classifier with lowest weighted non-conditional selectivity $\psi_1^\sigma/\mu_1^\sigma$. Determine $\begin{bmatrix} t_1^\sigma \\ g_1^\sigma \end{bmatrix}$.
- Calculate the ratio $\psi_2^\sigma/\mu_2^\sigma$ for all remaining $N-1$ classifiers. Select $C_{\sigma(2)}$ as the classifier with lowest weighted conditional selectivity $\psi_2^\sigma/\mu_2^\sigma$. Determine $\begin{bmatrix} t_2^\sigma \\ g_2^\sigma \end{bmatrix}$.
- Continue until all classifiers have been selected.

In each iteration we have to update $O(N)$ selectivities and there are $O(N)$ iterations, making the complexity of the algorithm $O(N^2)$ (compared to $O(N!)$ for the optimal algorithm). Yet, it can be shown that the performance of the Greedy Algorithm can be bound:

$$\frac{1}{\kappa} U_{ord}^{opt} \leq U_{ord}^{G} \leq U_{ord}^{opt} \qquad \text{with } \kappa = 4.$$

The value U_{ord}^{G} of the utility obtained with the Greedy Algorithm's order is at least 1/4th of the value of the optimal order U_{ord}^{opt}. Furthermore, the approximation factor $\kappa = 4$ corresponds to a system with infinite number of classifiers [34]. In practice, this constant factor is smaller. Specifically, we have $\kappa = 2.35, 2.61, 2.8$ for 20, 100 or 200 classifiers respectively.

The key of the proof of this result is to show that the Greedy Algorithm is equivalent to a greedy 4-approximation algorithm for pipelined set-cover. We refer the interested reader to the demonstration made by Munagala and Ali in [2] and let him show that our problem setting is equivalent to the one formulated in their problem.

3.3 Joint Order and Operating Point Selection

Further system performance can be achieved by both optimizing the order of the chain of classifiers and the operating point configuration.

To build a joint order and operating point selection strategy, we propose to combine the SQP-based solution for operating point selection with the iterative Greedy order selection. This iterative approach, or SQP-Greedy algorithm, is summarized as follows:

Centralized Algorithm 2 SQP-Greedy algorithm for joint ordering and operating point selection

- **Initialize** $\sigma^{(0)}$.
- **Repeat** until greedy algorithm does not modify order.
  1. Given order $\sigma^{(j)}$, compute locally optimal $\mathbf{x}^{(j)}$ through SQP.
  2. Given operating points $\mathbf{x}^{(j)}$, update order $\sigma^{(j+1)}$ using (A-)Greedy algorithm.

Each step of the SQP-Greedy algorithm is guaranteed to improve the global utility of the problem. Given a maximum bounded utility, the algorithm is then guaranteed to converge. However, it may be difficult to bound the performance gap between the SQP-Greedy and the optimal algorithm with a constant factor, since the SQP only achieves local optima. As a whole, identification and optimization of algorithms used to compute optimal order and operating points represents a major roadblock to stream mining optimization.

3.3.1 Limits of Centralized Algorithms for Order Selection

We want to underline that updating the *ex-post* selectivities requires strong coordination between classifiers. A first solution would be for classifiers to send their choice of operating point (p^F, p^D) to a central agent (which would also have knowledge about the *a-priori* conditional selectivities ϕ^σ, $\overline{\phi^\sigma}$) and would compute the *ex-post* conditional selectivities. A second solution would be for each classifier C_i to send their rates t_i and g_i to the classifiers C_j which have not yet processed the stream for them to compute ψ^i_j. In both cases, heavy message exchange is required, which can lead to system inefficiency (cf. Sect. 4.1). We will propose in Sect. 4 a decentralized solution with limited message exchanges, as an alternative to this centralized approach.

3.4 Multi-Chain Topology

3.4.1 Motivations for Using a Multi-Chain Topology: Delay Tradeoff Between Feature Extraction and Intra-Classifier Communication

In the previous analysis, we did not take into consideration the time α^{com} required by classifiers to communicate with each other. If classifiers are all grouped on a single node, such communication time α^{com}_{inter} can be neglected compared to the time α^{feat} required by classifiers to extract data features. However for classifiers instantiated on separate nodes, this communication time α^{com}_{ext} can greatly increase the total time required to deal with a stream tuple.

As such, we would like to limit the communication between nodes, i.e. (1) avoid sending the stream back and forth from one node to another and (2) limit message exchanges between classifiers. To do so, a solution would be to process the stream in parallel on each node and to intersect the output of each node-chain.

3.4.2 Number of Chains and Tree Configuration

Suppose that instead of considering classifiers in a chain, we process the stream through R chains, where chain r has N_r classifiers with the order σ_r. The answer of the query is then obtained by intersecting the output of each chain r (we assume that this operation incurs zero delay).

We can show that, as a first approximation, the end-to-end processing time can be written as

$$c^{\sigma}_{delay} = \underbrace{\sum_{r=1}^{R}\sum_{h=1}^{N_r} \alpha^{feat}_{\sigma_r(h)} t^{\sigma_r}_{h-1}}_{feature extraction} + \underbrace{\sum_{r=1}^{R}\sum_{h=1}^{N_r-1} \alpha^{com}_{\sigma_r(h),\sigma_r(h+1)} t^{\sigma_r}_{h}}_{intra-classifier communication}. \quad (8)$$

Intuitively, the feature extraction term increases with the number of chains R, as each chain needs to process the whole stream, while the intra-classifier communication term decreases with R, since using multiple chains enables classifiers instantiated on the same node to be grouped together in order to avoid time-costly communication between nodes (cf. Fig. 4b).

Configuring stream mining systems as tree topologies (i.e. determining the number of chains to use in order to process the stream in parallel, as well as the composition and order of each chain) represents a major research theme. The number of chains R and the choice of classifiers per chain illustrate the tradeoff between feature extraction and intra-classifier communication and will depend on the values of α^{feat} and α^{com}.

4 Decentralized Approach

4.1 Limits of Centralized Approaches and Necessity of a Decentralized Approach

The centralized approach presented in the previous sections has six main limitations:

1. **System and Information Bottlenecks:** Centralized approaches require a central agent that collects all information, generates optimal order and operating points per classifier, and distributes and enforces results on all classifiers. This creates a

bottleneck, as well as a single point of failure, and is unlikely to scale well as the number of classifiers, topologic settings, data rates, and computing infrastructure grow.

2. **Topology Specificity:** A centralized approach is designed to construct one topology for each user application of interest. In practice the system may be shared by multiple such applications—each of which may require the reuse of different subsets of classifiers. In this case, the centralized algorithm needs to design multiple orders and configurations that need to be changed dynamically as application requirements change, and applications come and go.
3. **Resource Constraints:** Currently designed approaches minimize a combination of processing delay and misclassification penalty. However, in general we also need to satisfy the resource constraints of the underlying infrastructure. These may in general lead to distributed non-convex constraints in the optimization, thereby further increasing the sub-optimality of the solution, and increasing the complexity of the approach.
4. **Synchronization Requirements:** The processing times vary from one classifier to the other. As a result, transmission from one classifier to another is not synchronized. Note that this asynchrony is intrinsic to the stream mining system. Designing one centralized optimization imposes synchronization requirements among classifiers and as the number of classifiers and the size of the system increases may reduce the overall efficiency of the system.
5. **Limited Sensitivity to Dynamics:** As an online process, stream mining optimization must involve algorithms which take into account the system's dynamics, both in terms of the evolving stream characteristics and classifiers' processing time variations. This time-dependency is all the more true in a multi-query context, with heterogeneous data streams for which centralized algorithms are unable to cope with such dynamics.
6. **Requirement for Algorithms to Meet Time Delay Constraints:** These dynamics require rapid adaptation of the order and operating points, often even at the granularity of one tuple. Any optimization algorithm thus needs to provide a solution with a time granularity finer than the system dynamics. Denote by τ the amount of time required by an algorithm to perform one iteration, i.e. to provide a solution to the order and configuration selection problem. The solution given by an algorithm will not be obsolete if $\tau \leq \mathfrak{C}\tau^{dyn}$ where τ^{dyn} represents the characteristic time of significant change in the input data and characteristics of the stream mining system and $\mathfrak{C} \leq 1$ represents a buffer parameter in case of bursts.

To address these limitations, we propose a decentralized approach and design a decentralized stream mining framework based on reinforcement learning techniques.

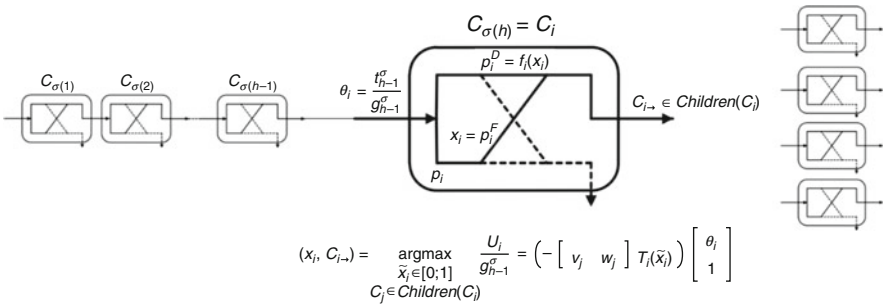

Fig. 10 Stochastic decision process: at each node, optimisation of local utilisation of select operating point and child classifier

4.2 Decentralized Decision Framework

The key idea of the decentralized algorithm is to replace centralized order selection by local decisions consisting in determining to which classifier to forward the stream. To describe this, we set up a stochastic decision process framework $\{\mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{U}\}$ [15], illustrated in Fig. 10, where

- $\mathcal{C} = \{C_1, \ldots, C_N\}$ represents the set of classifiers
- $\mathcal{S} = \underset{i \leq N}{\times} \mathcal{S}_i$ represents the set of states
- $\mathcal{A} = \underset{i \leq N}{\times} \mathcal{A}_i$ represents the set of actions
- $\mathcal{U} = \{U_1, \ldots, U_N\}$ represents the set of utilities

4.2.1 Users of the Stream Mining System

Consider N classifiers $\mathcal{C} = \{C_1, \ldots, C_N\}$. The classifiers are autonomous: unless otherwise mentioned, they do not communicate with each other and take decisions independently. We recall that the hth classifier will be referred as $C_i = C_{\sigma(h)}$. We will also refer to the stream source as $C_0 = C_{\sigma(0)}$.

4.2.2 States Observed by Each Classifier

The set of states can be decomposed as $\mathcal{S} = \underset{i \leq N}{\times} \mathcal{S}_i$. The local state set of classifier $C_i = C_{\sigma(h)}$ at the hth position in the classifier chain is defined as $\mathcal{S}_i = \{(Children(C_i), \theta_i)\}$:

- $Children(C_i) = \left\{C_k \in \mathcal{C} | C_k \notin \{C_{\sigma(1)}, C_{\sigma(2)} \ldots, C_i\}\right\} \subset \mathcal{C}$ represents the subset of classifiers through which the stream still needs to be processed after

it passes classifier C_i. This is a required identification information to be included in the header of each stream tuple such that the local classifier can know which classifiers still need to process the tuple.

- The throughput-to-goodput ratio $\theta_i = \frac{t^\sigma_{h-1}}{g^\sigma_{h-1}} \in [1, \infty]$ is a measure of the accuracy of the ordered set of classifiers $\{C_{\sigma(1)}, C_{\sigma(2)}, \ldots, C_i\}$. Indeed, $\theta_i = 1$ corresponds to perfect classifiers $C_{\sigma(1)}, C_{\sigma(2)}, \ldots, C_i$, (with $p^D = 1$ and $p^F = 0$), while larger θ_i imply that data has been either missed or wrongly classified.

  The state θ_i can be passed along from one classifier to the next in the stream tuple header. Since $\theta_i \in [1, \infty]$, the set of states $\mathcal{S}_i$ is of infinite cardinality. For computational reasons, we would require a finite set of actions. We will therefore approximate the throughput-to-goodput ratio by partitioning $[1, \infty]$ into L bins $S_l = [b_{l-1}, b_l]$ and approximate $\theta_i \in S_l$ by some fixed value $s_l \in S_l$.

4.2.3 Actions of a Classifier

Each classifier C_i has two independent actions: it selects its operating point x_i and it chooses among its children the trusted classifier $C_{i\rightarrow}$ to which it will transmit the stream. Hence $\mathcal{A}_i = \{(x_i, C_{i\rightarrow})\}$, where

- $x_i \in [0, 1]$ corresponds to the operating point selected by C_i.
- $C_{i\rightarrow} \in Children(C_i)$ corresponds to the classifier to which C_i will forward the stream. We will refer to $C_{i\rightarrow}$ as the trusted child of classifier C_i.

Note that the choice of trusted child $C_{i\rightarrow}$ is the local equivalent of the global order σ. The order is constructed classifier by classifier, each one selecting the child to which it will forward the stream: $\forall h \in [1, N]$, $C_{\sigma(h)} = C_{\sigma(h-1)\rightarrow}$.

4.2.4 Local Utility of a Classifier

We define the local utility of a chain of classifiers by backward induction:

$$U_{\sigma(h)} = -\rho_{\sigma(h)} t^\sigma_{h-1} + U_{\sigma(h+1)} \qquad \text{and} \qquad U_{\sigma(N)} = -\rho_{\sigma(N)} t^\sigma_{N-1} + g^\sigma_N - K t^\sigma_N. \tag{9}$$

The end-to-end utility of the chain of classifiers can then be reduced to $U = U_{\sigma(1)}$.

The key result of this section consists in the fact that the global optimum can be achieved locally with limited information. Indeed, each classifier $C_i = C_{\sigma(h)}$ will globally maximize the system's utility by autonomously maximizing its local utility $U_i = \underbrace{\left[\, v^\sigma_h \;\; w^\sigma_h \,\right]}_{=[v_i \;\; w_i]} \begin{bmatrix} t^\sigma_{h-1} \\ g^\sigma_{h-1} \end{bmatrix}$ where the local utility parameters $\left[\, v^\sigma_h \;\; w^\sigma_h \,\right]$ are defined recursively:

$$\begin{aligned}\left[\, v_N^\sigma \;\; w_N^\sigma \,\right] &= -\left[\, \rho_{\sigma(N)} \;\; 0 \,\right] + \left[\, -K \;\; 1 \,\right] T_N^\sigma \\ \left[\, v_h^\sigma \;\; w_h^\sigma \,\right] &= -\left[\, \rho_{\sigma(h)} \;\; 0 \,\right] + \left[\, v_{h+1}^\sigma \;\; w_{h+1}^\sigma \,\right] T_h^\sigma .\end{aligned}$$

This proposition can easily be proven recursively.

Therefore, the local utility of classifier C_i can now be rewritten as

$$U_i = \left(-\left[\, \rho_i \;\; 0 \,\right] + \left[\, v_{h+1}^\sigma \;\; w_{h+1}^\sigma \,\right] T_i^\sigma(x_i) \right) \begin{bmatrix} t_{h-1}^\sigma \\ g_{h-1}^\sigma \end{bmatrix}. \tag{10}$$

As such, the decision of classifier C_i only depends on its operating point x_i, on the state θ_i which it observes[5] and on the local utility parameters $\left[\, v_j \;\; w_j \,\right]$ of its children classifiers $C_j \in Children(C_i)$. Once it knows the utility parameters of all its children, classifier C_i can then uniquely determine its best action (i.e. its operating point x_i and its trusted child $C_{i\rightarrow}$) in order to maximize its local utility.

4.3 Decentralized Algorithms

At this stage, we consider classifiers with fixed operating points. The action of a classifier C_i is therefore limited to selecting the trusted child $C_{i\rightarrow} \in Children(C_i)$ to which it will forward the stream.

4.3.1 Exhaustive Search Ordering Algorithm

We will say that a classifier C_i *probes* a child classifier C_j when it requests its child utility parameters $\left[\, v_j \;\; w_j \,\right]$.

To best determine its trusted child, a classifier only requires knowledge on the utility parameters of all its children. We can therefore build a recursive algorithm as follows: all classifiers are probed by the source classifier C_0; to compute their local utility, each of the probed classifiers then probes its children for their utility parameters $\left[\, v \;\; w \,\right]$. To determine these, each of the probed children needs to probe its own children for their utility parameter, etc. The local utilities are computed in backwards order, from leaf classifiers to the root classifier C_0. The order yielding the maximal utility is selected.

Observe that this decentralized ordering algorithm leads to a full exploration of all $N!$ possible orders at each iteration. Achieving the optimal order only requires one iteration, but this iteration requires $O(N!)$ operations and may thus

[5] t_{i-1} and g_{i-1} are not required since: $\text{argmax}\; U_i = \text{argmax}\; \frac{U_i}{g_{i-1}} = \left(-\left[\, \rho_i \;\; 0 \,\right] + \left[\, v_{i+1} \;\; w_{i+1} \,\right] T_i^\sigma \right) \begin{bmatrix} \theta_i \\ 1 \end{bmatrix}$.

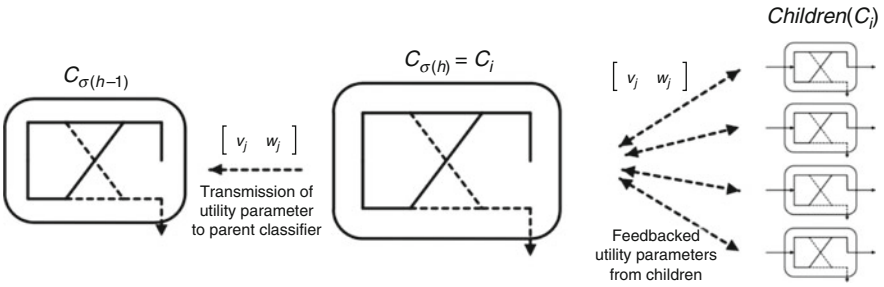

Fig. 11 Feedback information for decentralized algorithms

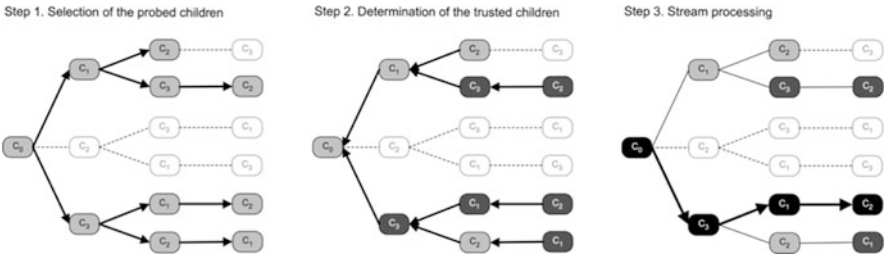

Fig. 12 Global Partial Search Algorithm only probes a selected subset of classifier orders

require substantial time, since heavy message exchange is required (Fig. 11). For quasi-stationary input data, the ordering could be performed offline and such computational time requirement would not affect the system's performance. However, in bursty and heterogeneous settings, we have to ensure that the optimal order calculated by the algorithm would not arrive too late and thus be completely obsolete. In particular, the time constraint $\tau \leq \mathfrak{C}\tau^{dyn}$, defined in Sect. 4.1 must not be violated.

We therefore need algorithms capable of quickly determining a good order, though convergence may require more than one iteration. In this way, it will be possible to reassess the order of classifiers on a regular basis to adapt to the environment.

4.3.2 Partial Search Ordering Algorithm

The key insight we want to leverage is to screen only through a selected subset of the $N!$ orders at each iteration. Instead of probing all its children classifiers systematically, the hth classifier will only request the utility parameters $\left[\ v\ w\ \right]$ of a subset of its $N - h$ children.

From a *global* point of view, one iteration can be decomposed in three major steps, as shown on Fig. 12:

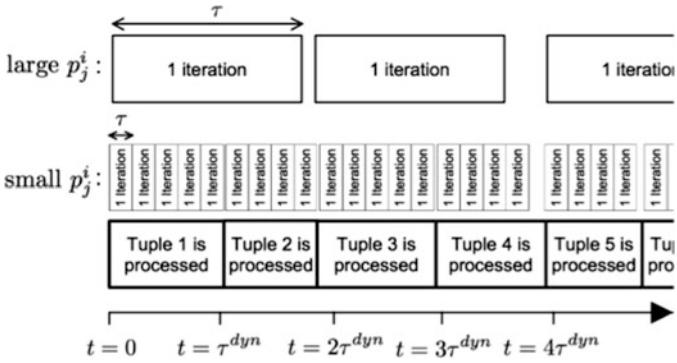

Fig. 13 Time scales for decentralized algorithms

Step 1: Selection of the Children to Probe A partial tree is selected recursively (light grey on Fig. 12). A subset of the N classifiers are probed as first classifier of the chain. Then, each of them selects the children it wants to probe, each of these children select the children which it wants to probe, etc.

Step 2: Determination of the Trusted Children The order to be chosen is determined backwards: utilities are computed from leaf classifiers to the source classifier C_0 based on feedback utility parameters. At each node of the tree, the child classifier which provides its parent with the greatest local utility is selected as the trusted child (dark grey on Fig. 12).

Step 3: Stream Processing The stream is forwarded from one classifier to its trusted child (black on Fig. 12).

If we want to describe Step 1 more specifically, classifier C_i will probe its child C_j with probability p_j^i. As will be shown in Sect. 4.5, adjusting the values of p_j^i will enable to adapt the number of operations and the time τ required per iteration, as shown on Fig. 13. Indeed, for low values of p_j^i, few of the $N!$ orders will be explored, and since each classifier only probes a small fraction of its children, one iteration will be very rapid. However, if the values of p_j^i are close to 1, each iteration requires a substantial amount of probing and one iteration will be long.

In the Partial Search Ordering Algorithm, one classifier may appear at multiple depths and positions in the classifiers' tree. Each time, it will realize a *local* algorithm described in the flowchart in Fig. 14.

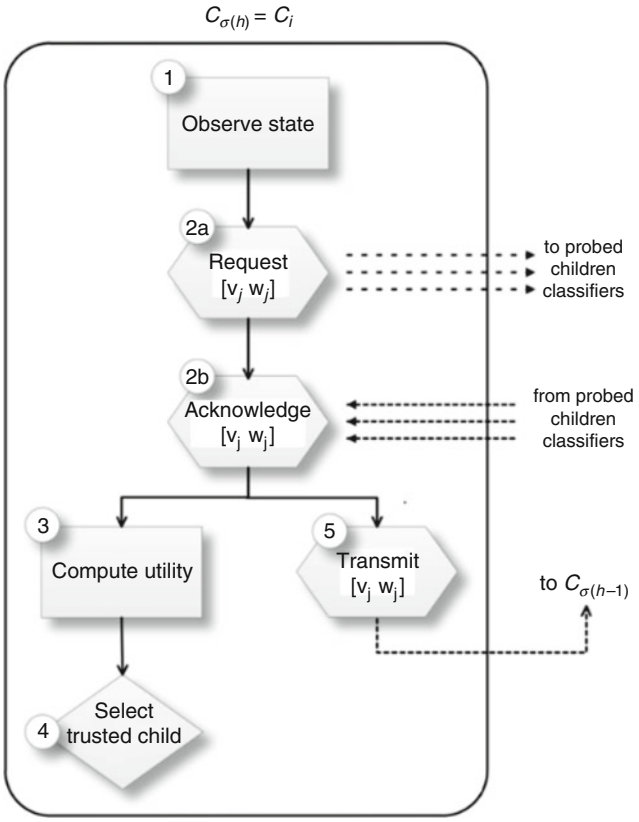

Fig. 14 Flowchart of local algorithm for partial search ordering

Decentralized Algorithm 3 Partial Search Ordering Algorithm—*for classifier* $C_i = C_{\sigma(h)}$

1. **Observe state** $(\theta_i, Children(C_i))$
2. With probability p_j^i, **request utility parameters** $\left[\, v_{\sigma(h+1)} \;\; w_{\sigma(h+1)} \,\right] = \left[\, v_j \;\; w_j \,\right]$ for any of the $N - h$ classifiers $C_j \in Children(C_i)$
3. For each child probed, **compute** corresponding **utility**

$$U_i(C_j) = \left(-\left[\, \rho_{\sigma(i)} \;\; 0 \,\right] + \left[\, v_j \;\; w_j \,\right] T_i^{:0}\right) \begin{bmatrix} t_{h-1}^{\sigma} \\ g_{h-1}^{\sigma} \end{bmatrix}$$

4. **Select** the child classifier with the highest U_i as **trusted child**.
5. Compute the corresponding $\left[\, v_i \;\; w_i \,\right]$ and **transmit** it to a previous classifier who requested it.

4.3.3 Decentralized Ordering and Operating Point Selection

In case of unfixed operating points, the local utility of classifier $C_i = C_{\sigma(h)}$ also depends on its local operating point x_i—but it does not directly depend on the operating points of other classifiers[6]:

$$U_i = \left(- \left[\rho_i \ 0 \right] + \left[v_{h+1}^{\sigma} \ w_{h+1}^{\sigma} \right] T_i^{\sigma}(x_i) \right) \begin{bmatrix} t_{h-1}^{\sigma} \\ g_{h-1}^{\sigma} \end{bmatrix}.$$

As a consequence, we can easily adapt the Partial Search Ordering Algorithm into a Partial Search Ordering and Operating Point Selection Algorithm by computing the maximal utility (in terms of x_i) for each child:

$$U_i(C_j) = \max_{x_i} \left(- \left[\rho_{\sigma(i)} \ 0 \right] + \left[v_j \ w_j \right] T_i^{\sigma}(x_i) \right) \begin{bmatrix} t_j \\ g_j \end{bmatrix}. \tag{11}$$

To solve the local optimization problem defined in Eq. (11), each classifier can either derive the nullity of the gradient if the ROC curve function $f_i : p^F \mapsto p^D$ is known, or search for optimal operating point using a dichotomy method (since $U_i(C_j)$ is concave).

4.3.4 Robustness of the Partial Search Algorithm and Convergence Speed

It can be shown that under stable conditions the Partial Search Algorithm converges and the equilibrium point of the stochastic decision process. For fixed operating point the Partial Search Algorithm converges to the optimal order if $p_j^i > 0 \ \forall \ i, j$.

In case of joint ordering and operating point selection, there exist multiple equilibrium points, each corresponding to a local minimum of the utility function. The selection of the equilibrium point among the set of possible equilibria depends on the initial condition (i.e. order and operating points) of the algorithm. To select the best equilibrium, we can perform the Partial Search Algorithm for multiple initial conditions and keep only the solution which yielded the maximum utility.

In practice, stable stream conditions will not be verified by the stream mining system, since the system's characteristics vary at a time scale of τ^{dyn}. Hence, rather than achieving convergence, we would like the Partial Search Algorithm to reach near-equilibrium fast enough for the system to deliver solution to the accuracy and delay joint optimization on a timely basis.

In analogy to [9], we first discuss how model-free Safe Experimentation, a heuristic case of Partial Search Algorithm can be used for decentralized stream mining and leads to a low-complexity algorithm, however with slow convergence

[6]The utility parameters $\left[v_j \ w_j \right]$ fed back from classifier C_j to classifier C_i are independent of any classifiers' operating points.

rate. Fortunately, the convergence speed of the Partial Search Algorithm can be improved by appropriately selecting the screening probabilities p_j^i. In Sect. 4.5, we will construct a model-based algorithm which enables to control the convergence properties of the Partial Search Algorithm, and lead to faster convergence.

4.4 Multi-Agent Learning in Decentralized Algorithm

We aim to construct an algorithm which would maximize as fast as possible the global utility of the stream mining system expressed in Eq. (4). We want to determine whether it is worthwhile for a classifier C_i to probe a child classifier C_j for its utility parameters and determine search probabilities p_j^i of the Partial Search Algorithm accordingly.

4.4.1 Tradeoff Between Efficiency and Computational Time

Define an experiment $E_{i\to j}$ as classifier C_i's action of probing a child classifier C_j by requesting its utility parameter $\left[\, v_j \;\; w_j \,\right]$. Performing an experiment can lead to a higher utility, but will induce a cost in terms of computational time:

- Denote by $\hat{U}(E_{i\to j}|s_k)$ the expected additional utility achieved by the stream mining system if the experiment $E_{i\to j}$ is performed under state s_k.
- Let τ^{ex} represent the expected amount of time required to perform an experiment. This computational time will be assumed independent of the classifiers involved in the experiment performed and the state observed.

Then, the total expected utility per iteration is given by $\hat{U}(p_j^i) = \sum p_j^i \hat{U}(E_{i\to j}|s_k)$ and the time required for one iteration is $\tau(p_i^j) = \hat{n}(p_j^i)\tau^{ex}$, where $\hat{n}(p_j^i)$ represents the expected number of experiments performed in one iteration of the Partial Search Algorithm and will be defined precisely in the next paragraph.

The allocation of the screening probabilities p_j^i aims to maximize the total expected utility within a certain time:

$$\begin{cases} \underset{p_j^i \in [0,1]}{\text{maximize}} \; \hat{U}(p_j^i) \\ \text{subject to } \tau(p_i^j) \leq \mathfrak{C}\tau^{dyn} \end{cases}. \tag{12}$$

4.4.2 Safe Experimentation

We will benchmark our results on Safe Experimentation algorithms as cited in [9]. This low-complexity, model-free learning approach was first proposed for large-

scale, distributed, multi-agent systems, where each agent is unable to observe the actions of all other agents (due to informational or complexity constraints) and hence cannot build a model of other agents [17]. The agent therefore adheres to a "trusted" action at most times, but occasionally "explores" a different one in search of a potentially better action.

Safe Experimentation is a reinforcement learning algorithm where each classifier learns by observing the payoff with which its past actions were rewarded. As such, it does not consider the interactions between agents, or in our case, the actions of other autonomous classifiers. In particular, there is no explicit message exchange among classifiers required (i.e. no $\left[v \; w \right]$ exchanged), though each classifier needs to know the reward of its action (and computing this reward might yet require some form of implicit communication between agents).

The Safe Experimentation algorithm is initialized by selecting an initial order σ_0 of classifier. $C_{\sigma_0(h+1)}$ will be referred as the "trusted" child of the hth classifier $C_{\sigma_0(h)}$. At each time slot, $C_{\sigma(h)}$ will either forward the stream to its "trusted" child $C_{\sigma(h+1)}$ with probability $(1 - \varepsilon)$ or, with probability ε, will explore a classifier C_j chosen randomly among its children. In the case where a higher reward is achieved through exploration, C_j will become the new "trusted" child of $C_{\sigma(h)}$. Note that so long as $\epsilon > 0$, all possible orders will ultimately be explored, such that Safe Experimentation converges to the optimal order [9].

Instead of considering a fixed exploration rate ϵ, we can consider a diminishing exploration rate ϵ_t. In this way, the algorithm will explore largely for first iterations and focus on exploited orders near convergence. $\epsilon_t \rightarrow 0$ and $\prod_{t=1}^{\infty} \left(1 - \frac{{\epsilon_t}^{N-1}}{(N-1)!} \right) \rightarrow 0$ are sufficient conditions for convergence, typically verified for $\epsilon_t = (1/t)^{1/n}$.

Two majors limits of Safe Experimentation can be identified:

- **Slow convergence**: One iteration of Safe Experimentation is very rapid ($O(N)$), since only one order is experienced. However, the expected number of iterations required to converge to optimal order is bounded below by $N!$ (corresponding to uniform search: $\epsilon_t = 1$). As a consequence, the time required to reach the optimal solution might be infinitely long, since the optimal order could be experimented after an infinitely large number of iterations.
- **General approach**: This slow convergence can be explained by the fact that Safe Experimentation, as a model-free approach, does not leverage the structure of the problem studied. In particular, one major constraint fixed by Safe Experimentation is to try only one classifier among all its children, while the stream mining optimization problem allows to probe multiple children simultaneously by requesting their utility parameters $\left[v \; w \right]$ and selecting the trusted child based on these fed back values. This capacity to try multiple orders per iterations will enable to build a parameterized algorithm to speed-up the convergence to optimal order by choosing screening probabilities p^i_j appropriately.

4.5 Parametric Partial Search Order and Operating Point Selection Algorithm

As expressed in (12), the screening probabilities p^i_j can be used to tradeoff the expected utility and the computational cost. In this final section, we frame a general methodology aiming to determine the optimal tradeoff. In order to be adaptable to the setting considered, we construct our learning algorithm in three steps, each step representing a certain granularity level, and each step being controllable by one "macroscopic" state variable. Doing so, we put forward three tradeoffs corresponding to three independent questions: (1) how much to search?, (2) how deep to search?, (3) where to search?

This enables the construction of a parametric learning algorithm, extensively detailed in [8]. This article shows that the probability $p^i_j(p, \xi, \beta)$ that the hth classifier $C_{\sigma(h)} = C_i$ probes its children C_j, given that it received data with throughput-to-goodput ratio $\theta_i \in S_k$ can be expressed as:

$$p^i_j(p, \xi, \beta) = \underbrace{p}_{howmuch?} \times \underbrace{\frac{C'}{1 + e^{-\frac{h - [Np]}{\xi}}}}_{howdeep?} \times \underbrace{\frac{e^{\beta U_i(j,k)}}{\sum_{C_l \in Children(C_i)} e^{\beta U_i(l,k)}}}_{where?}.$$

The reader will find a justification of the formalism of p^i_j in [8].

4.5.1 Controlling the Screening Probability

Using this expression of p^i_j is meant to be able to control key characteristics of the screening probability by tuning parameters p,ξ and β.

The first parameter $p = \underset{i,j}{Av}(p^i_j)$ is used to arbitrate between rapid but inflexible search and slower but system-compliant search. Its value will impact the time τ required for one iteration and has to be selected small enough in order to ensure that $\tau \leq \mathfrak{C}\tau^{dyn}$, thus, coping with environment dynamics.

The second control parameter ξ is used to arbitrate between rapid but less secure search and slower but exhaustive search. It is a refinement parameter, which dictates how much more extensive search should be performed in the lower classifiers than in the upper ones.

- $\xi = 0$ corresponds to searching only for last classifiers and violates the exhaustivity of the search (no optimal convergence ensured).
- $0 < \xi < \infty$ corresponds to searching more exhaustively for last classifiers than for first classifiers.
- $\xi = \infty$ corresponds to searching uniformly at any depth with probability p.

The third parameter β balances the options of probing unexplored children versus exploiting already-visited orders, by weighting the propensity of classifier $C_i = C_{\sigma(h)}$ to probe one of its children classifier C_j, based on past experiments' reward. In most learning scenarios where the tradeoff between exploration and exploitation arises, both exploitation and exploration cannot be performed at the same time: the algorithm will either exploit well-known tracks or explore new tracks [14]. This is due to the absence of immediate feedback. In our setting, each classifier requests the information (i.e. utility parameters) of a subset of its children and is then able to base its decision on their feedback information.

In practice, the weight associated to a specific child based on its past reward could be determined using any increasing function f. Using $f_\beta(U) = \frac{e^{\beta U}}{\sum_V e^{\beta V}}$ is motivated by the analogy of a classifiers utility U to an energy [22]. In this case, $\frac{e^{\beta U}}{\sum_V e^{\beta V}}$ represents the equilibrium probability of being at an energy level U. As such, the parameter β can be interpreted as the inverse of a temperature, i.e. it governs the amount of excitation of the system:

- $\beta = 0$ corresponds to a very excited system with highly time-varying characteristics. In this case, since characteristics change very quickly, random exploration: $p^i_j = p^i$ is recommended by the algorithm.
- $\beta = \infty$ corresponds to a non-varying system. Then, full-exploitation of past rewards is recommended (given that all states were explored at least once) and weight should be concentrated only on the child which provides the maximum utility.
- $0 < \beta < \infty$ is a tradeoff between exploration ($\beta = 0$) and exploitation ($\beta = \infty$) and corresponds to settings where algorithmic search and environment evolution are performed at the same time scale.

4.5.2 Comparison of Ordering and Operating Point Selection Algorithms

Our preliminary results in Table 1 compare the performance of several joint ordering and operating point selection algorithms based on important criteria in the considered stream mining system.

4.5.3 Order Selected by Various Classifiers for Different Ordering Algorithms

The performance of the different ordering algorithms are shown in Table 2 for seven classifiers with fixed operating points per classifier. The classifier's characteristics (p^F, p^D) (i.e. the ROC curve), ψ (i.e. the *ex-post* selectivities), and α (i.e. the resource requirements) were generated randomly. The misdetection cost $c^M = 10$,

Table 1 Comparison of ordering and operating point selection algorithms

Ordering and operating point selection algorithm	System compliance	Utility achieved	Message exchange	Speed of convergence	Adaptability	Control
SQP-Greedy	Low	Bound; local opt.	Heavy	Medium	Little	∅
Safe experimentation	High	Local opt.	∅	Medium	∅	∅
Partial search	Complete	Local opt.	Light	Rapid	Total	Yes

Table 2 Utilities and computational time achieved for different ordering algorithms

	Algorithm	Order obtained	Utility	Comp. time
Centralized	Optimal	[C_6 C_2 C_1 C_4 C_3 C_7 C_5]	100	>5 min
	Greedy	[C_4 C_1 C_6 C_7 C_2 C_3 C_5]	95	0.002 s
Decentralized	Safe experimentation	[C_6 C_2 C_1 C_4 C_3 C_7 C_5]	100	2.09
	Partial search	[C_6 C_2 C_1 C_4 C_3 C_7 C_5]	100	1.2 s

false alarm cost $c^F = 1$, and $\lambda = 0.1$. The input data rate $t_0 = g_0$ was selected to normalize the optimal utility to 100.

As expected, the globally optimal centralized solution requires too much computation time, while the centralized Greedy algorithm does not lead to the optimal order, but results in very little computational time. The Parametric Partial Search Algorithm (here with $p = 0, 1$, $T = 1$ and $\beta = 0$) converges quicker than Safe Experimentation (here with $\epsilon = 0, 1$), to the optimal order. Decentralized algorithms converge to the optimal order, given that they ultimately probe all the possible orders, but they require longer computational time than the centralized greedy solutions. However, as shown in [8], convergence to a near-optimal order requires only a few iterations.

5 Online Learning for Real-Time Stream Mining

Mining dynamic and heterogeneous data streams using optimization tools may not always be feasible due to the unknown and time-varying distributions of these streams. Since classification of the streaming data needs to be done immediately, and invoking a classifier is costly, choosing the right classifier at run time is an important problem. In this section we review numerous methods that learn which classifiers to invoke based on the streaming meta-data, which is also called the context. All the algorithms studied in this section are able to mine big data streams in real-time. To accomplish this task, they are designed to have the following key properties:

- After a data instance is classified, the result is used to update the parameters of the learning algorithms. Then, the data instance is discarded. Therefore, it is not necessary to keep the past data instances in the memory.
- Usually, a single classifier is selected to classify the current data instance. As a result, only the accuracy estimate for the selected classifier is updated. While all the algorithms discussed in this section are capable of simultaneously updating the accuracies of all of the classifiers, this can lead to significant computational overhead due to the fact that it requires predictions from all of the classifiers for each data instance. Nevertheless, the performance bounds discussed in this section also holds for the case above.

5.1 Centralized Online Learning

This subsection is devoted to the study of centralized online learning algorithms for stream mining. We introduce several challenges related to real-time stream mining, various performance measures and the algorithms that address each one of the introduced challenges.

5.1.1 Problem Formulation

Each classifier takes as input a data instance and outputs a prediction. The data stream also includes a stream of meta-data, which is also referred to as the context stream. At time t, the data instance $s(t)$ is observed together with the context $x(t) \in \mathcal{X}$. The label $y(t) \in \{0, 1\}$ is not observed. The context can be categorical, real-valued and/or multi-dimensional. For instance, in a medical diagnosis application, the data instance can be an MRI image of a tissue, while the context can be resolution, type of the scanner, age of the patient and/or radius of the tumor. Depending on the particular application, the set of all possible contexts $\mathcal{X}$ can be very large and even infinite. The dimension of the context space is denoted by D. Each dimension of the context is called a *context type*. For instance, for the medical diagnosis example given above "age" is a context type, while the specific value that this type takes is the context.

No statistical assumptions are made on the context stream. However, the data and the label is assumed to be drawn from a fixed distribution given the context. This departs from the majority of the supervised learning literature, which assumes that the data is i.i.d. over time. Based on this, the accuracy of a classifier C given context x is defined to be $\pi_C(x) \in [0, 1]$. The classifier accuracies are not known beforehand and need to be learned online.

It is common to assume that the accuracy of a classifier is similar for similar contexts [25, 30]. For instance, in a social network users with similar age, income and geographic location will have a tendency to click on similar ads, which will result in a similar accuracy for a classifier that tries to predict the ad that the user

will click to. This assumption, which is also called the *similarity assumption*, is mathematically formalized as Hölder continuity of the accuracy of classifier c as a function of the context:

$$|\pi_C(x) - \pi_C(x')| \leq L \times \text{dist}(x, x')^{\alpha} \tag{13}$$

where L is the Hölder constant, α is the Hölder exponent and $\text{dist}(\cdot, \cdot)$ is a distance metric for the contexts. For most of the cases α is set to be 1, which makes the accuracy of classifier f Lipschitz continuous in the context [24].

The standard performance measure for online learning is the regret, which is defined as

$$\text{Reg}(T) := \sum_{t=1}^{T} \pi^*(x(t)) - \text{E}\left[\sum_{t=1}^{T} \pi_{a(t)}(x(t))\right] \tag{14}$$

where $\pi^*(x(t)) := \max_{C \in \mathcal{C}} \pi_C(x(t))$ and $a(t)$ denotes the classifier selected at time t. Hence, minimizing the regret is equivalent to selecting the best classifier as many times as possible. The time-averaged regret is defined as $\overline{\text{Reg}}(T) = \text{Reg}(T)/T$. $\overline{\text{Reg}}(T) \to 0$ implies that the average performance is (asymptotically) as good as the average performance of the best classifier selection policy given the complete knowledge of classifier accuracies. In order for the time-averaged regret to converge to zero, the regret must grow at most sublinearly over time, i.e., $\text{Reg}(T) \leq K T^{\gamma}$ for some constants $K > 0$ and $\gamma \in [0, 1)$ for all T.

5.1.2 Active Stream Mining

Online learning requires knowing the labels, in order to update the accuracy of the selected classifier. In most of the stream mining applications, such as medical diagnosis, acquiring the label is costly. Hence, a judicious mechanism that decides when to acquire the label based on the confidence on the accuracy of the selected classifier needs to be developed. The performance measure, i.e., the regret, also needs to be re-defined to capture the cost of label acquisition; hence, it becomes

$$\text{Reg}(T) := \sum_{t=1}^{T} \pi^*(x(t)) - \text{E}\left[\sum_{t=1}^{T} \pi_{a(t)}(x(t)) - Jr(t)\right] \tag{15}$$

where $r(t)$ is 1 if label is acquired at time t and 0 otherwise, and J is a constant that represents the tradeoff between accuracy and label acquisition cost.

Since the number of possible contexts is usually very large, it is very inefficient to learn the classifier accuracies for each context separately. Therefore, the learning algorithms developed for stream mining learn the classifier accuracies for groups of similar contexts, where the groups are formed by partitioning the context space based on the similarity assumption given in (13). Then, the estimated accuracy of

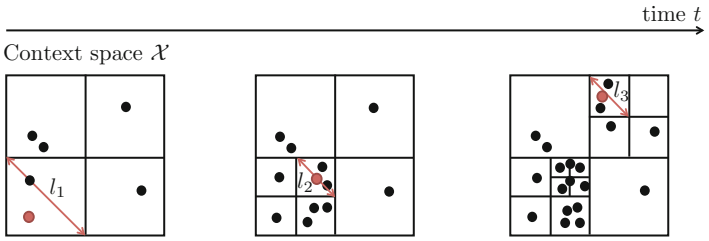

Fig. 15 Evolution of context space partition over time for $D = 2$. Red dots represent the most recent contexts and black dots represent the past contexts for which the label was acquired at the time of decision. Based on the similarity assumption, the type 2 error is proportional to l_j^{α}, $j = 1, 2, 3$, where l_j denotes the *diameter* of the group that the most recent context belongs to. On the other hand, the type 1 error decreases with the number of dots which belong to the group (square) that the current context belongs to

classifier C for context $x(t)$ is computed as $\hat{\pi}_C(x(t)) := \hat{\pi}_C(p(t))$ where $p(t)$ is the group that contains $x(t)$ in the partition of the context space. Here, $\hat{\pi}_C(p(t))$ is the sample mean of the correct predictions averaged over all past context arrivals to $p(t)$ for which the label was acquired. As shown in Fig. 15, this partition is adapted based on how the contexts arrive in order to balance the two sources of error in estimating the classifier accuracies: (1) type 1 error that arises from the number of past labeled data instances belonging to a group; (2) type 2 error that arises from the dissimilarity of the contexts that belong to the same group.

The label acquisition decision is also made to balance this tradeoff. Specifically, each label acquisition decreases the type 1 error of the selected classifier for the group that the current context belongs to. If accuracy of the selected classifier is known with a high confidence, then label acquisition is not necessary. In order to achieve this, the learning algorithm indefinitely alternates between two phases: exploration phase and exploitation phase, which are described below.

- **Exploration phase**: Select a classifier that the algorithm has a low confidence on its accuracy. After performing classification by the selected classifier, acquire the label of the data instance and update the accuracy estimate of the selected classifier.
- **Exploitation phase**: Select the classifier with the highest estimated accuracy, i.e., $a(t) = \arg\max_{C \in \mathcal{C}} \hat{\pi}_C(x(t))$.

After an exploration phase, the confidence on the accuracy of the selected classifier increases. Thus, classifier accuracies are learned in exploration phases. On the other hand, in an exploitation phase the prediction accuracy is maximized by classifying the data instance based on the empirically best classifier. In [25], it is shown that sublinear in time regret can be achieved by acquiring labels only sublinearly many times. While this regret bound holds uniformly over time, its dependence on T can be captured by using the asymptotic notation, which implies that $\mathrm{Reg}(T) = \tilde{O}(T^{(\kappa+D)/(\kappa'+D)})$, for some constants $\kappa' > \kappa > 0$. The specific

implementation keeps a control function $D(t)$, and explores only when the number of times the label is acquired by time t is less than or equal to $D(t)$. $D(t)$ increases both with t and the inverse of the diameter of the group that $x(t)$ belongs to. For this algorithm, the number of groups in the partition of the context space is also a sublinear function of time, which implies that the memory complexity of the algorithm is also sublinear in time. Moreover, identifying both the group that the current context belongs to and the empirically best classifier are computationally simple operations, which makes this algorithm suitable for real-time stream mining.

5.1.3 Learning Under Accuracy Drift

Since the data stream is dynamic, its distribution conditioned on the context can also change over time. This is called the concept drift [35]. It is straightforward to observe that the concept drift will also cause a change in the accuracy of the classifiers. For this setting, the time-varying accuracy of a classifier C for context x is denoted by $\pi_C(t, x)$. It is assumed that the accuracy *gradually drifts* over time, which can be written as

$$|\pi_C(t, x) - \pi_C(t', x')| \leq L \times \text{dist}(x, x')^{\alpha} + \frac{|t - t'|}{T_s}$$

where T_s denotes the *stability* of the concept. If T_s is large the drift is slow, while if T_s is small the drift is fast. Note that this assumption does not introduce any explicit restrictions on the data stream distribution. Hence, the accuracy drift is more general than the concept drift and can also model scenarios in which there is a change in the classifiers. For instance, in an application with SVM classifiers, some of the classifiers might be re-trained on-the-fly as more data instances arrive, which will result in a change in their decision boundaries, and hence their accuracies, even though the stream distribution remains the same.

An algorithm that learns and tracks the best classifier when there is accuracy drift is proposed in [26]. This algorithm estimates the classifier accuracies by using a recent time window of observations from similar contexts as opposed to using the entire past history of observations. In this work, the optimal window size is computed to be a sublinear function of T_s.

In general, it is not possible to achieve sublinear in time regret when there is accuracy drift due to the fact that the classifier accuracies are continuously changing. A constant rate of exploration is required in order to track the best classifier. A suitable performance measure for this setting is the time-averaged regret $\overline{\text{Reg}}(T)$. The algorithm proposed in [26] achieves a time-averaged regret of $\tilde{O}(T_s^{-\gamma})$, where $\gamma \in (0, 1)$ is a parameter that depends on α and the dimension of the context space. This implies that the time-averaged regret decreases as T_s increases, which is expected since it is easier to learn the classifier accuracies when the drift is slow.

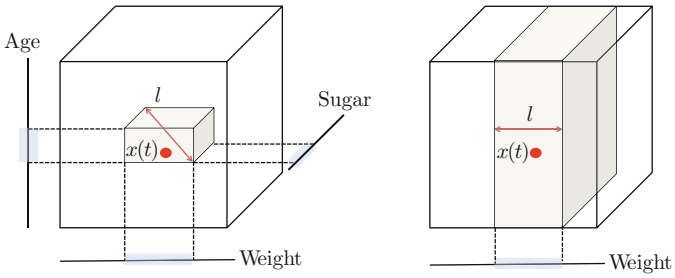

Fig. 16 A medical diagnosis example with three dimensional context space. Shaded areas represent the set of past observations that are used to estimate the classifier accuracies for the current context $x(t)$. On the left figure all context types are relevant, while on the right figure only the context type "weight" is relevant. The shaded areas on the left and right figures have the same type 2 error. However, the type 1 error for the shaded area on the right figure is much less than the one on the left figure since it includes many more past observations

5.1.4 Learning the Relevant Contexts

When the dimension D of the context space is large, the methods proposed in the previous sections which rely on partitioning the context space suffer from the curse of dimensionality as shown in Fig. 16. As a result, the regret bound given in Sect. 5.1.2 becomes almost linear in time.

It is possible to avoid the curse of dimensionality when the classifier accuracies depend only on a small subset of the set of all possible context types. In stream mining, this implies that there are many irrelevant context types which do not affect the outcome of the classification.[7] If the relevant context types were known, online learning could be easily performed by partitioning the context space restricted to the relevant context types. However, identifying these relevant context types on-the-fly without making any statistical assumptions on how the contexts arrive is a challenging task. Nevertheless, it is possible to identify the relevant context types through a sophisticated *relevance test*. This test identifies relevance assumptions that are consistent with the classifier accuracies estimated so far. The only requirements for this test are (13) and an upper bound on the number of relevant context types.

Here, we explain the relevance test that identifies one relevant context type for every classifier. The extension to more than one relevant context type can be found in [28]. It is important to note that the relevance test is only performed in exploitation phases as it requires confident accuracy estimates. First, for each context type i, the variation of the estimated accuracy of classifier C over all pairs of context types that include context type i is calculated. The resulting vector is called the *pairwise variation vector* of context type i. Then, a bound on the variation of the estimated accuracy of classifier C due to type 2 errors, which is called *natural variation*, is

[7] The definition of irrelevant context types can be relaxed to include all context types which have an effect that is less than $\epsilon > 0$ on the classifier accuracies.

calculated. The set of candidate relevant context types are identified as the ones for which the pairwise variation is less than a linear function of the natural variation. Finally, a context type i^* is selected from the set of candidate relevant types to be the estimated relevant context type, and the accuracies of the classifiers are re-calculated by averaging over all past observations whose type i^* contexts are *similar* to the current type i^* context. In [29] it is shown that online learning with relevance test achieves regret whose time order does not depend on D (but depends on the number of relevant context types). Hence, learning is fast and efficient, given that the number of relevant context types is much smaller than D.

5.2 Decentralized Online Learning

In this subsection, we consider how online learning can be performed in distributed classifier networks. We review two methods: cooperative contextual bandits in which local learners (LLs) cooperate to learn the best classifier to select within the network; hedged bandits in which an ensemble learner (EL) fuses the predictions of the LLs to maximize the chance of correct classification.

5.2.1 Problem Formulation

Most of the definitions and notations are the same as in Sect. 5.1.1. There are M data streams, each of which is processed locally by its LL. Each LL has a set of classifiers $\mathcal{C}_i$ that it can use to classify its incoming data stream. The set of all classifiers is denoted by $\mathcal{C} = \cup_{i=1}^{M} \mathcal{C}_i$. The context that is related to the ith data stream is denoted by $x_i(t)$, where $t \in \{1, 2, \ldots\}$.

5.2.2 Cooperative Contextual Bandits

In this part we describe a cooperative online learning framework that enables an LL to use other LLs' classifiers to classify its own data stream. For cooperative contextual bandits, the assumption on the context arrival process is the same as the assumption in Sect. 5.1.1.

In order to understand the benefit of cooperation, first we consider the case in which each LL acts individually. In this case, the highest accuracy that LL i can get for context x is $\pi_i^*(x) := \max_{C \in \mathcal{C}_i} \pi_C(x)$. On the other hand, if all LLs cooperate and share their classifiers with each other, the highest accuracy LL i can get for context x is $\pi_G^*(x) := \max_{C \in \mathcal{C}} \pi_C(x)$. Clearly, $\pi_G^*(x) \geq \pi_i^*(x)$. However, it is not straightforward for LLs to achieve a classification accuracy that is equal to $\pi_G^*(x)$ for all $x \in \mathcal{X}$ due to the following reasons:

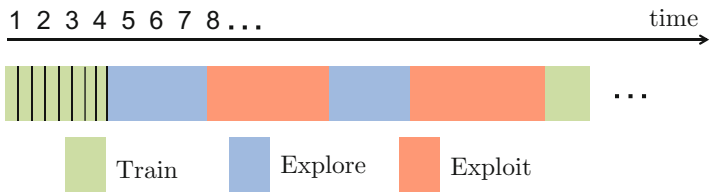

Fig. 17 Interleaving of the training, exploration and exploitation phases over time for a particular LL and context arrival process in cooperative contextual bandits

- LL i does not known the accuracies of its classifiers $\pi_C(x)$, $C \in \mathcal{C}_i$, $x \in \mathcal{X}$ a priori, and needs to learn these accuracies on-the-fly.
- LL i does not know the classifiers available to the other LLs. Moreover, other LLs may be reluctant to share such an information due to privacy constraints.
- LL i cannot observe the data streams arriving to other LLs due to both privacy and communication constraints. However, LL i is able to send selected data instances to other LLs (possibly by incurring some communication cost) and is able to receive selected data instances from the other LLs.

The challenging decentralized learning problem stated above is solved by designing a decentralized learning algorithm that promotes cooperation among the learners [27]. In this algorithm, each learner alternates between three different phases over time as given in Fig. 17. The sequencing of these phases is adapted online based on the context arrival process. In each phase the learning algorithm takes an *action* for a different purpose:

- **Training phase**: LL i selects another LL and sends its context and data instance to the selected LL. Then, the selected LL is asked to classify the data instance. After the classification is performed and the true label is received by LL i, this true label is also send to the selected LL in order for it to update the accuracy of the classifier that it had selected on behalf of LL i. Hence, the purpose of the training phase is to train other LLs such that they learn the accuracies of their own classifiers with a high confidence for contexts that arrive to LL i.
- **Exploration phase**: LL i selects one of its own classifiers or another LL for the purpose of learning the accuracy.
- **Exploitation phase**: LL i selects one of its own classifiers or another LL for the purpose of maximizing the probability of correct classification.

Due to the heterogeneity of the data streams, usually it is not possible for a single LL to learn its classifier accuracies well for all possible contexts by just observing its own data stream. This can happen because a context that is rare for one LL can be frequent for another LL. While this results in asymmetric learning, it is solved by the training phase.

Note from Fig. 17 that exploration is performed only when there is no need for training. This is to ensure that if another LL is selected to make a classification, it performs classification based on its best classifiers. Otherwise, LL i might learn the

accuracy of the other LLs incorrectly, which might results in failure to identify an LL, whose accuracy is higher than the accuracies of the classifiers in $\mathcal{C}_i$. Similarly, exploitation is performed only when there is no need to train any other LL or to explore any other LL or classifier.

One important question is how much training and exploration is needed. This can be analytically solved by defining confidence intervals for the sample mean (empirical) estimates of the classifier accuracies, and adjusting these confidence intervals over time to achieve a certain performance goal. In cooperative contextual bandits, the regret of LL i is defined as

$$\mathrm{Reg}_i(T) := \sum_{t=1}^{T} \pi^*_G(x_i(t)) - \mathrm{E}\left[\sum_{t=1}^{T} \pi_{a_i(t)}(x_i(t))\right] \tag{16}$$

where $\pi_{a_i(t)}(x_i(t))$ denotes the accuracy of the classifier selected by LL $a_i(t)$ on behalf of LL i for $a_i(t) \notin \mathcal{C}_i$. Again, we seek to achieve sublinear in time regret, which implies that the learning algorithm's average number of correct predictions converges to that of the $\pi^*_G(x)$.

Specifically, it is proven in [27] that sublinear regret can be achieved by an algorithm that uses sublinear number of training and exploration phases. In order to achieve sublinear regret, the classifier accuracies must also be learned together for similar contexts by a context space partitioning method such as the one given in Fig. 15.

5.2.3 Hedged Bandits

Hedged Bandits model decentralized stream mining applications in which all data streams are related to the same event. Hence, it is assumed that the contexts, data instances and labels are drawn according to an i.i.d. process. LLs produce predictions by choosing classifiers according to their own learning algorithms, classifiers and data streams, and then, send these predictions to an EL, which fuses the predictions together to produce a final prediction. The learning algorithm used by the LLs is similar to the learning algorithms discussed in Sect. 5.1. On the other hand, the EL uses a variant of the Hedge algorithm [10] that does not require the time horizon to be known in advance. This guarantees that the ELs prediction is as good as the predictions of the best LL given the context [30].

6 Conclusion

Adapting in real time the topology of classifiers and their configuration (operating point) enables to significantly improve the performance of stream mining systems, by optimally trading up accuracy and delay, under limited resources.

However, the emergence of new stream mining applications, with specific requirements and characteristics, widens the spectrum of possibilities and leaves room for further improvements. Today, more and more data is available to be processed, and more and more classification, filtering, analysis or sorting can be performed on this data. As such, a major challenge lies in identifying, prioritizing and planning mining tasks. Until now, the mapping between queries and a set of corresponding classifiers was considered as given. Yet, this mapping should be decided jointly with the topology construction and the system configuration for an optimal stream mining design.

An upstream consideration would be to decide whether streams should be systematically classified or only identified upon request. Indeed, a stream mining system must not only be seen as a query-centric processing system aiming to identify which subset of data answers a given set of queries. Instead of defining the set of classifiers on the basis of the set of queries ($\mathcal{C} = \bigcup_{q \in \mathcal{Q}} \mathcal{C}(q)$), we could determine what are all the queries which can be answered given a set of classifiers ($\mathcal{Q} = \{q \mid \mathcal{C}(q) \subset \mathcal{C}\}$). Since such classifier-centric approach leads to an explosion of the number of queries which can be processed (N binary classifiers can potentially process $\sum_{k=1}^{N} \frac{N!}{k!(N-k)!} 2^k$ different queries) it is critical to be able to identify or to learn online which classification to perform.

Indeed, classifier design is an expensive process and determining which feature to extract represents a major topic in the data mining community. Hence, given a data stream and a query, deciding which classifiers should match which queries has not yet been analytically studied. Underlying this issue resonates the exploration versus exploitation tradeoff, where we need to train the stream mining system to detect the classifiers which are critical to stream identification.

Acknowledgements This work is based upon work supported by the National Science Foundation under Grant No. 1016081. We would like to thank Dr. Deepak Turaga (IBM Research) for introducing us to the topic of stream mining and for many productive conversation associated with the material of this chapter as well as providing us with Figs. 1 and 3 of this chapter. We also would like to thank Dr. Fangwen Fu and Dr. Brian Foo, who have been PhD students in Prof. van der Schaar group and have made contributions to the area of stream mining from which this chapter benefited. Finally, we thank Mr. Siming Song for kindly helping us with formatting and polishing the final version of the chapter.

References

1. Babcock, B., Babu, S., Datar, M., Motwani, R.: Chain: Operator scheduling for memory minimization in data stream systems. In: Proc. ACM International Conference on Management of Data (SIGMOD), pp. 253–264 (2003)
2. Babu, S., Motwani, R., Munagala, K., Nishizawa, I., Widom, J.: Adaptive ordering of pipelined stream filters. In: ACM SIGMOD International Conference on Management of Data (2004)
3. Boyd, S.P., Vandenberghe, L.: Convex Optimization. Cambridge University Press (1994)
4. Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y., Zdonik, S.: Scalable distributed stream processing. In: Proc. of Conference on Innovative Data Systems Research, Asilomar (2003)

5. Cherniack, M., Balakrishnan, H., Carney, D., Cetintemel, U., Xing, Y., Zdonik, S.: Scalable distributed stream processing. In: Proc. CIDR (2003)
6. Condon, A., Deshpande, A., Hellerstein, L., Wu, N.: Flow algorithm for two pipelined filter ordering problems. In: ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (2006)
7. Douglis, F., Branson, M., Hildrum, K., Rong, B., Ye, F.: Multi-site cooperative data stream analysis. ACM SIGOPS **40**(3) (2006)
8. Ducasse, R., Turaga, D.S., van der Schaar, M.: Adaptive topologic optimization for large-scale stream mining. IEEE Journal on Selected Topics in Signal Processing **4**(3), 620–636 (2010)
9. Foo, B., van der Schaar, M.: Distributed classifier chain optimization for real-time multimedia stream-mining systems. In: Proc. IS&T / SPIE Multimedia Content Access, Algorithms and Systems II (2008)
10. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: Proc. European Conference on Computational Learning Theory, pp. 23–37 (1995)
11. Fu, F., Turaga, D.S., Verscheure, O., van der Schaar, M., Amini, L.: Configuring competing classifier chains in distributed stream mining systems. IEEE Journal on Selected Topics in Signal Processing (2007)
12. Gaber, M., Zaslavsky, A., Krishnaswamy, S.: Resource-aware knowledge discovery in data streams. In: Proc. First Intl. Workshop on Knowledge Discovery in Data Streams (2004)
13. Garg, A., Pavlovic, V.: Bayesian networks as ensemble of classifiers. In: Proc. 16th International Conference on Pattern Recognition (ICPR), pp. 779–784 (2002)
14. Gupta, A., Smith, K., Shalley, C.: The interplay between exploration and exploitation. Academy of Management Journal (2006)
15. Hu, J., Wellman, M.: Multiagent reinforcement learning: Theoretical framework and an algorithm. In: Proceedings of the Fifteenth International Conference on Machine Learning (1998)
16. Low, S., Lapsley, D.E.: Optimization flow control I: Basic algorithm and convergence. IEEE/ACM Trans. Networking **7**(6), 861–874 (1999)
17. Marden, J., Young, H., Arslan, G., Shamma, J.: Payoff based dynamics for multi-player weakly acyclic games. SIAM Journal on Control and Optimization, special issue on Control and Optimization in Cooperative Networks (2007)
18. Merugu, S., Ghosh, J.: Privacy-preserving distributed clustering using generative models. In: Proc. of 3rd International Conference on Management of Data, pp. 211–218 (2003)
19. Olston, C., Jiang, J., Widom, J.: Adaptive filters for continuous queries over distributed data streams. In: Proc. ACM SIGMOD Intl. Conf. Management of Data, pp. 563–574 (2003)
20. Palomar, D., Chiang, M.: On alternative decompositions and distributed algorithms for network utility problems. In: Proc. IEEE Globecom (2005)
21. Park, H., Turaga, D.S., Verscheure, O., van der Schaar, M.: Foresighted tree configuring games in resource constrained distributed stream mining systems. In: Proc. IEEE Int. Conf. Acoustics Speech and Signal Process. (2009)
22. Saul, L., Jordan, M.I.: Learning in Boltzmann trees. Neural Computation (1994)
23. Schapire, Y.: A brief introduction to boosting. In: Proc. International Conference on Algorithmic Learning Theory (1999)
24. Slivkins, A.: Contextual bandits with similarity information. Journal of Machine Learning Research **15**(1), 2533–2568 (2014)
25. Tekin, C., van der Schaar, M.: Active learning in context-driven stream mining with an application to image mining. IEEE Transactions on Image Processing **24**(11), 3666–3679 (2015)
26. Tekin, C., van der Schaar, M.: Contextual online learning for multimedia content aggregation. IEEE Transactions on Multimedia **17**(4), 549–561 (2015)
27. Tekin, C., van der Schaar, M.: Distributed online learning via cooperative contextual bandits. IEEE Transactions Signal Processing **63**(14), 3700–3714 (2015)

28. Tekin, C., van der Schaar, M.: RELEAF: An algorithm for learning and exploiting relevance. IEEE Journal of Selected Topics in Signal Processing **9**(4), 716–727 (2015)
29. Tekin, C., Van Der Schaar, M.: Discovering, learning and exploiting relevance. In: Advances in Neural Information Processing Systems, pp. 1233–1241 (2014)
30. Tekin, C., Yoon, J., van der Schaar, M.: Adaptive ensemble learning with confidence bounds. IEEE Transactions on Signal Processing **65**(4), 888–903 (2017)
31. Turaga, D., Verscheure, O., Chaudhari, U., Amini, L.: Resource management for networked classifiers in distributed stream mining systems. In: Proc. IEEE ICDM (2006)
32. Vaidya, J., Clifton, C.: Privacy-preserving k-means clustering over vertically partitioned data. In: Proc. of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 206–215 (2003)
33. Varshney, P.: Distributed Detection and Data Fusion. Springer (1997). ISBN: 978-0-387-94712-9
34. Vazirani, V.: Approximation Algorithms. Springer Verlag, Inc., New York, NY, USA (2001)
35. Žliobaitė, I.: Learning under concept drift: an overview. arXiv preprint arXiv:1010.4784 (2010)