Distributed Online Learning and Stream Processing for a Smarter Planet

Deepak S. Turaga, Mihaela van der Schaar

Abstract

Smarter Planet applications for transportation, healthcare, energy and utilities have been enabled by the confluence of technological advancements in sensing, communication, and compute systems, and analysis algorithms. These applications have have several novel characteristics in terms of their distributed nature, their analysis, security and privacy needs, their performance scaling needs, and their dynamics – that have not been sufficiently addressed by prior centrally-store-process-and-analyze systems and algorithms.

In this chapter we describe the emerging field of Stream Processing systems and its impact on the realization of these applications – and discuss how this is enabling fog computing in real-world implementations. We introduce the paradigm, its core constructs and capabilities, and examples of academic, commercial and open-source systems available for use. We also highlight examples of large-scale distributed applications built on these systems, and in use in current Smarter Planet applications. We then discuss how these systems enable several new directions of research, at the intersection of online learning, application-system design and optimization.

We illustrate one such research direction by describing several novel online distributed learning algorithms that can be implemented on a stream processing system to provide solutions to real-time prediction problems in the presence of diverse data sources as well as missing and/or delayed data or feedback. We show how such distributed algorithms can then be used for real-world prediction problems in social networks, transportation networks and healthcare informatics. We also include pointers to other interesting research in this space. We conclude with our thoughts on the open challenges in this space, and the likely evolution of such systems and algorithms.

I. Introduction: Smarter Planet

With the world becoming ever more instrumented and connected, we are at the cusp of realizing a *Smarter Planet* [1], where insights drawn from data sources are used to adapt our environment and how we interact with it and with each other. This will enable a range of new services that

make it easier for us to work, travel, consume, collaborate, communicate, play, entertain, and even be provided care.

Consider the pervasiveness of the mobile phone. It is rapidly emerging as the primary digital device of our times - with over 6 (out of the 7) billion people in the world having access to a mobile phone [2]. We are witnessing the rapid emergence of services that use these phones (especially smart phones) as sensors of the environment and interfaces to people. For instance, it is now common with several map services (e.g., Google Maps) to be provided a live view of the traffic across a road network. This aggregate view is computed by processing and analyzing in real-time the spatio-temporal properties of data collected from several millions of mobile phones. Applications such as Waze include adding crowd-sourced information to such data, where individual people use mobile phones to report traffic congestion, accidents, etc., and these are then transmitted to other users to inform and potentially alter their travel. While several of these applications are focused on aggregate information processing and dissemination, it is natural to expect more personalized applications, including *personal trip advisors*, that can provide dynamic routing, as well as potentially combine multi-modal transport options (e.g., car-train-walk-bus).

Cities, that are responsible for providing several transportation related services, can use information from mobile phones, augmented with their own road sensors (loop sensors, cameras, etc.) and transport sensors (GPS on buses and trains, etc.), to optimize their transport grid in real-time, provide emergency services (e.g., evacuations and dynamic closures), real-time toll, modify public transport (e.g., allow for dynamic connections between bus/train routes based on current demand), and even control their traffic light systems. This ecosystem, including individual consumers and city infrastructure, is shown in Fig. 1.

These types of applications have several unique requirements in terms of streaming data management and communication, preprocessing and cleaning, analysis and mining, scaling, and finally adaptation. Computing for these applications has to be distributed end-to-end, from the user devices all the way to the cloud, requiring the Fog Computing paradigm. There need to be several advances in sensing and communication technology coupled with development of new analytic algorithms and platforms for these individual centric and city-wide applications to become real, and deliver value[1]. In this chapter, we introduce the emerging paradigm of *Stream*

[1]While our description has been focused on transportation applications, there are several applications of these types in different domains ranging from healthcare, financial services, physical and cyber security, telecommunications, energy and utility, and environmental monitoring.

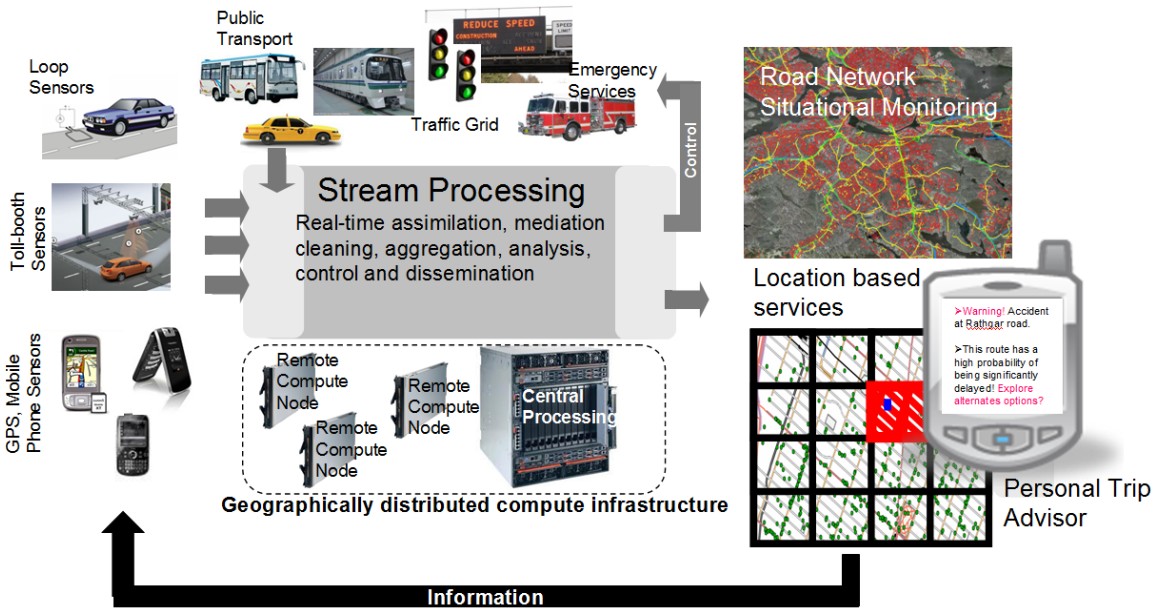

Fig. 1: Smarter transportation: individuals and city.

Processing and Analysis, including novel platforms and algorithms, that support the requirements of these kinds of applications. We introduce distributed stream processing systems, and propose a novel distributed online learning framework that can be deployed on such systems to provide a solution to an illustrative smarter planet problem. We believe that the recent arrival of new *freely available* systems for distributed stream processing such as InfoSphere Streams [3], Storm [4] and Spark [5], enable several new directions for advancing the state-of-the-art in large-scale, real-time analysis applications, and provide the academic and industrial research community the tools to devise end-to-end solutions to these types of problems, and overcome issues with proprietary or piecemeal solutions.

This chapter is organized as follows. We start by defining a specific real-world transportation inspired problem that requires large-scale online learning, in Section II. We then formalize the characteristics of such problems and their associated challenges in Section III. We discuss distributed systems in Section IV, and how the emergence of stream processing systems allows us to build and deploy appropriate solutions to these problems. Following this, in Section V, we propose a new framework for distributed, online, ensemble learning that can naturally be deployed on a stream processing system to realize such applications, and we describe how to apply such a framework to a collision detection application. We conclude with a discussion on the several directions for future research enabled by this combination, in Section VI.

II. Illustrative Problem: Transportation

In this section we define a concrete illustrative problem related to our transportation application domain that requires a joint algorithm-system design for online learning and adaptation. Consider a scenario where a city wants to modify its digital signage based on real-time predictions (e.g., 10 min in advance) of congestion in a particular zone. A visual depiction of this example is included in Fig. 2, where the white spot – at the intersection of major road links – is the point of interest for congestion prediction.

Data for this real-time prediction can be gathered from many different types of sensors. In this example we consider cell-phone location information from different points of interest, and local weather information. This data is naturally distributed and may not be available at one central location, due to either geographical diversity, or different cell phone providers owning different subsets of the data (i.e., their customers). In this simple example we consider geographic distribution of the data. The congestion prediction problem then requires deploying multiple distributed predictors that collect data from local regions and generate *local predictions*, that are then merged to generate a more reliable *final prediction*.

An example with two distributed predictor applications – each depicted as a flowgraph – is shown in Fig. 2. The two different flowgraphs in this example look at different subsets of the data, and implement appropriate operations for preprocessing (cleaning, denoising, merging, alignment, spatio-temporal processing, etc.), followed by operations for learning and adaptation to compute the local prediction. These are shown as the subgraphs labeled Pre-proc and Learner, respectively. In the most general case, a collection of different models (e.g., neural networks, decision trees, etc.), trained on appropriate training data, can be used by each predictor application.

The learners receive delayed feedback about their prediction correctness after a certain amount of time (e.g., if the prediction is for 10 minutes in advance, the label is available after 10 minutes) and can use it to modify their individual models and the local aggregation. Additionally, these learners also need to exchange information about their predictions across distributed locations, so that they can get a more global view of the state of the network and can improve their predictions. Prediction exchange between the learners is shown on the figure using dashed lines. Finally, the predictions from the learners can be used to update digital signage in real-time and potentially alert or divert traffic as necessary.

We formalize the characteristics of this online data stream processing application in the next section, and discuss how developing it requires the design of online, distributed ensemble learning

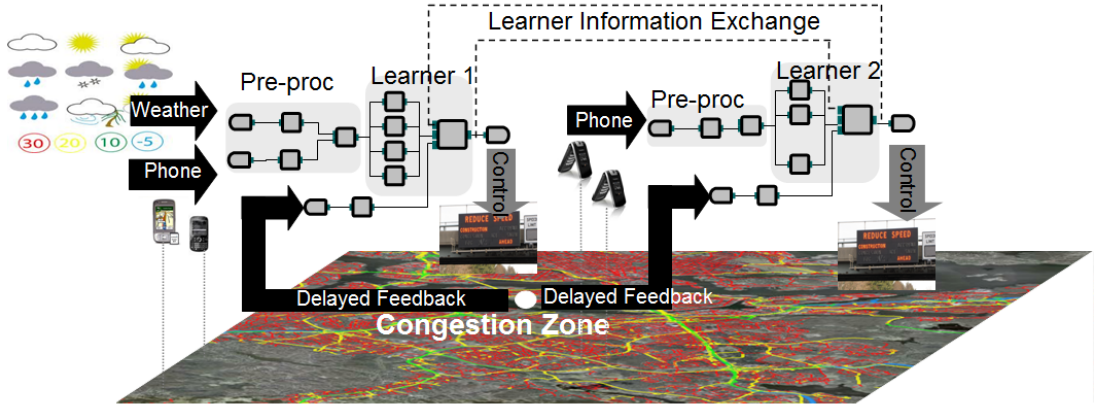

Fig. 2: Distributed learning needed for real-time signage update.

frameworks, while deploying it requires being able to instantiate these frameworks on a distributed system. In the following two sections, we show how we can leverage characteristics of modern stream processing systems in order to build and deploy general learning frameworks that solve distributed learning problems of this type. Our intent is to showcase how this enables a whole new way of thinking about such problems and opens up several avenues for future research.

III. Stream Processing Characteristics

There is a unique combination of multiple features that distinguishes Stream Processing Applications (SPAs) from traditional data analysis paradigms, which are often batch and offline. These features can be summarized as follows:

Streaming and In-Motion Analysis (SIMA): SPAs need to process streaming data on-the-fly, as it continues to flow, in order to support real-time, low-latency analysis, and to match the computation to the naturally streaming properties of the data. This limits the amount of prior data that can be accessed, and necessitates one-pass, online algorithms [6]–[8]. Several streaming algorithms are described in [9], [10].

Distributed Data and Analysis (DDA): SPAs analyze data streams that are often *distributed*, and their large rates make it impossible to adopt centralized solutions. Hence, the applications themselves need to be distributed.

High Performance and Scalable Analysis (HPSA): SPAs require high *throughput*, low *latency*, and dynamic *scalability*. This means that SPAs should be structured to exploit distributed computation infrastructures and different forms of parallelism (e.g., pipelined data and tasks). This

also means that they often require joint application and system optimization [11], [12].

Multi-Modal Analysis (MMA): SPAs need to process streaming information across heterogeneous data sources, including *structured* (e.g., transactions), *unstructured* (e.g., audio, video, text, image), and *semi-structured* data. In our transportation example this includes sensor readings, user contributed text and images, traffic cameras, etc.

Loss-Tolerant Analysis (LTA): SPAs need to analyze lossy data with different noise levels, statistical and temporal properties, mismatched sampling rates, etc., and hence they often need appropriate processing to transform, clean, filter and convert data and results. This also implies the need to match data rates, handle lossy data, synchronize across different data streams, and handle various protocols [7]. SPAs need to account for these issues and provide graceful degradation of results to loss in the data.

Adaptive and Time-varying Analysis (ATA): SPAs are often long-running and need to adapt over time to changes in the data and problem characteristics. Hence, SPAs need to support dynamic reconfiguration based on feedback, current context, and results of the analysis [6]–[8].

Systems and algorithms for SPAs need to provide capabilities that address these features and their combinations effectively.

IV. Distributed Stream Processing Systems

The signal processing and research community has so far focused on the theoretical and algorithmic issues for the design of SPAs, but has had limited success in taking such applications into real world deployments. This is primarily due to the multiple practical considerations involved in building an end-to-end deployment, and the lack of a comprehensive system and tools that provide them the requisite support. In this section, we summarize efforts at building systems to support SPAs and their shortcomings, and then describe how current stream processing systems can help realize such deployments.

A. State of the art

Several systems, combining principles from data management and distributed processing, have been developed over time to support different subsets of requirements that are now central to SPAs. These systems include traditional databases and data warehouses, parallel processing frameworks, active databases, continuous query systems, publish-subscribe (pub-sub) systems, Complex Event Processing (CEP) systems, and more recently the Map-Reduce frameworks. A quick summary of

System	SIMA	DDA	HPSA	MMA	LTA	ATA
Databases (DB2, Oracle, MySQL)	No	Yes	Partly	No	Yes	No
Parallel Processing (PVM, MPI, OpenMP)	No	Yes	Yes	Yes	Yes	No
Active Databases (Ode, HiPac, Samos)	Partly	Partly	No	No	Yes	Yes
Continuous Query Systems (NiagaraCQ, OpenCQ)	Partly	Partly	No	No	Yes	Yes
Pub-Sub Systems (Gryphon, Siena, Padres)	Yes	Yes	No	No	Yes	Partly
CEP Systems (Esper, SASE, WBE, Tibco BE, Oracle CEP)	Yes	Partly	Partly	No	Yes	Partly
Map-Reduce (Hadoop)	No	Yes	Yes	Yes	Yes	No

TABLE I: Data management systems and their support for SPA requirements: Streaming and In-Motion Analysis (SIMA), Distributed Data and Analysis (DDA), High-Performance and Scalable Analysis (HPSA), Multi-Modal Analysis (MMA), Loss Tolerant Analysis (LTA) and Adaptive and Time-varying Analysis (ATA).

the capabilities of these systems with respect to the streaming application characteristics defined in Section III is included in Table I.

More details on the individual systems and their examples can be obtained from [9]. However, as is clear, none of these systems were truly designed to handle all requirements of SPAs. As a consequence, this is an urgent need to develop more sophisticated stream processing systems.

B. Stream Processing Systems

While Stream Processing Systems (SPS) were developed by incorporating ideas from these preceding technologies, they required several advancements to the state-of-the-art in algorithmic, analytic, and systems concepts. These advances include sophisticated and extensible programming models, allowing continuous, incremental, and adaptive algorithms, and distributed, fault-tolerant, and enterprise-ready infrastructures or runtimes. These systems are designed to allow end-to-end distribution of real-time analysis, as needed in a fog computing world. Examples of early SPSs include TelegraphCQ, STREAM, Aurora, Borealis, Gigascope, Streambase [9]. Currently available and widely used SPSs include IBM InfoSphere Streams [3] (Streams), and the open-source Storm [4] and Spark [5] platforms. These platforms are freely available for experimentation and usage in commercial, academic, and research settings. These systems have been extensively deployed in multiple domains for telecommunication call detail record analysis, patient monitoring in ICUs, social media monitoring for real-time sentiment extraction, monitoring of large manufacturing systems (e.g. semiconductor, oil and gas) for process control, financial

services for online trading and fraud detection, environmental and natural systems monitoring etc. Descriptions of some of these real-world applications can be found in [9]. These systems have also been shown to scale to rates of millions of data items per second, deployed across 10s of thousands of processors in a distributed cluster, provide latencies of microseconds or lower, and connect with millions of distributed sensors.

While we omit a detailed description of stream processing platforms, we illustrate some of their core capabilities and constructs to support the needs of SPAs by outlining an implementation of the transportation application described in Section II) in a Stream Programming Language (SPL). In Fig. 2, the application is shown as a flowgraph which captures logical flow of data from one processing stage to another. Representing an application as a flowgraph allows for modular design and construction of the implementation, and as we discuss later, it allows stream processing systems to optimize the deployment of the application onto distributed computational infrastructures.

Each node on the processing flowgraphs in Fig. 2 that consumes and/or produces a stream of data is labeled an *operator*. Individual streams carry data items or tuples that can contain structured numeric values, unstructured text, semi-structured content such as XML, as well as binary blobs. In Table II we present an outline implementation in SPL [3]. Note that this flowgraph actually implements the distributed learning framework that will be discussed in more detail in Section V.

In this code, logical composition is indicated by an operator instance `stream <type> S3 = MyOp(S1;S2)`, where operator `MyOp` consumes streams `S1` and `S2` to produce stream `S3`, where type represents the type of tuples on the stream. Note that these streams may be produced and consumed on different computational resources – but that is transparent to the application developer. Systems like Streams also include multiple operators/tools that are required to build such an application. Examples include operators for:

- Data Sources and Connectors, e.g., `FileSource`, `TCPSource`, `ODBCSource`
- Relational Processing, e.g., `Join`, `Aggregate`, `Functor`, `Sort`
- Time Series Analysis, e.g., `Resample`, `Normalize`, `FFT`, `ARIMA`, `GMM`
- Custom Extensions, e.g., user created operators in C++/Java, or wrapping for Matlab, Python and R code.

These constructs allow for the implementation of distributed and ensemble learning techniques, such as those introduced in the learning framework, within specialized operators. We discuss this

```
composite Learner1 {
graph
 stream <TPhone> PhoneStream = TCPSource() {param role: server; port: 12345u;}
 stream <TWeather> WeatherStream = InetSource() {param URIList: ["http://noaa.org/xx"];}
 stream <TPhone> CleanPhoneStream = Custom(PhoneStream){...}
 stream <TWeather> CleanWeatherStream = Custom(WeatherStream){...}
 stream <TFeature> FeatureStream = Join(CleanWeatherStream; CleanPhoneStream){...}
 stream <TPrediction> P1 = MySVM(FeatureStream){...}
 stream <TPrediction> P2 = MyNN(FeatureStream){...}
 stream <TPrediction> Learner1Pred = MyAggregation(P1;P2;Feedback;Learner2Pred){...}
 () as Sink2 = Export(Learner1Pred){param properties: {name = "Learner1"}}
 stream <TPrediction> Learner2Pred = Import(){param properties: {name = "Learner2"}}
 stream <TFeedback> Feedback = Import(){param properties: {name = "Feedback"}}
}
```

TABLE II: Example of Streams programming to realize congestion prediction application flowgraph.

more in Section V. Stream processing platforms also include special tools for geo-spatial processing (e.g., for distance computation, map-matching, speed and direction estimation, bounding-box calculations) and standard mathematical processing. A more exhaustive list is available from [3].

Finally, programming constructs like the `Export` operator allows applications (in our case the congestion prediction application) to publish their output stream such that other applications, e.g., other learners, can dynamically connect to receive these results. This construct allows for dynamic connections to be established and torn down as other learners are instantiated, or choose to communicate. This is an important requirement for the learning framework in Section V.

These constructs allow application developers to focus on the core algorithm design and logical flowgraph construction, while the system provides support for communication of tuples across operators, conversion of the flowgraph into a set of processes or Processing Elements (PEs), distribution and placement of these PEs across a distributed computation infrastructure, and finally necessary optimizations for scaling and adapting the deployed applications. We present an illustration of the process used by such systems to convert a logical flowgraph to a physical deployment in Fig. 3.

Among the biggest strengths of using a stream processing platform is the natural scalability and efficiency provided by these systems, and their support for extensibility in terms of optimization techniques for *topology construction*, *operator fusion*, *operator placement*, and *adaptation* [9]. These systems allow users to formulate and provide additional algorithms to optimize their applications based on the application, data and resource specific requirements. This opens up several new research problems related to the joint optimization of algorithms and systems.

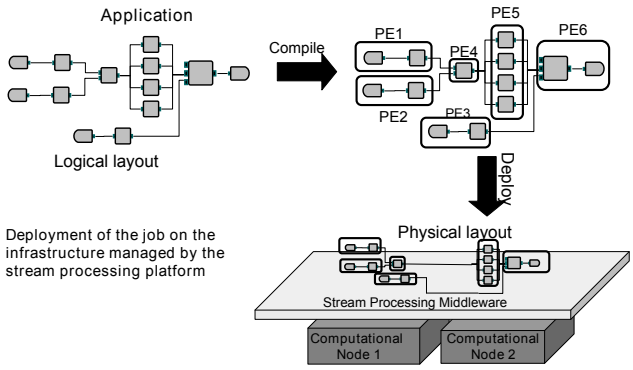

Fig. 3: From logical operator flowgraphs to deployment.

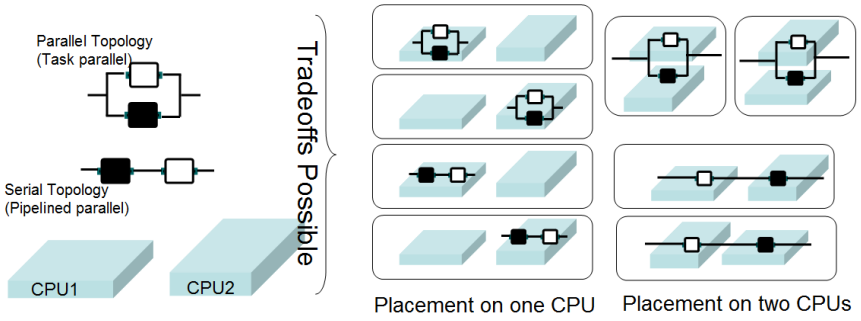

Fig. 4: Possible tradeoffs with parallelism and placement.

For instance, consider a simple example application with two operators. Using the Streams composition language, these operators – shown as black and white boxes in Fig. 4 – can be arranged into a parallel (task parallel) or a serial (pipelined parallel) topology to perform the same task[2], as shown in Fig. 4. This topology can also be distributed across computational resources (shown as different sized CPUs in Fig. 4). The right choice of topology and placement depends on the resource constraints and data characteristics, and needs to be dynamically adapted over time. This requires solving a joint resource optimization problem whose solution can be realized using the controls provided by stream processing systems. In practice, there are several such novel optimization problems that can be formulated and solved – in this joint application-system

[2]Note that with a stream processing platform other forms of parallelism such as data parallel processing and hybrid forms of parallelism with arbitrary combinations of data-, task-, and pipelined-parallel processing are possible.

research space. For instance, [13] and [11] discuss topology construction and optimization for non-trivial compositions of operators for multi-class classification. Additionally, these systems provide support for design of novel meta-learning and planning based approaches to dynamically construct, optimize and compose topologies of operators on these systems [14].

This combination of systems and algorithms enables several other open research problems in this space of joint application-system research, especially in an online, distributed, large-scale setting. In the next section we propose a solution to build a large-scale online distributed ensemble learning framework that leverages the capabilities provided by stream processing systems (and is implemented by the code in Table II) to provide solutions to the illustrative problem defined in Section II.

V. Distributed Online Learning Frameworks

We now formalize the problem described in Section II and propose a novel distributed learning framework to solve it. We first review the state of the art in such research and illustrate its shortcomings. Then we describe a systematic framework for online, distributed, ensemble learning well suited for SPAs. Finally, as an illustrative example, we describe how such framework can be applied to a collision detection application.

A. State of the art

As mentioned in Section III, it is important for stream processing algorithms to be online, one-pass, adaptive, distributed, to operate effectively under budget constraints, and to support combinations (or ensembles) of multiple techniques. Recently, there has been research that uses the above-mentioned techniques for analysis, and we include a summary of some of these approaches next.

Ensemble techniques [15] build and combine a collection of base algorithms (e.g., classifiers) into a joint unique algorithm (classifier). Traditional ensemble schemes for data analysis are focused on analyzing stored or completely available datasets; examples of these techniques include bagging [16] and boosting [17]. In the past decade much work has been done to develop online versions of such ensemble techniques. An online version of Adaboost is described in [18], and similar proposals are made in [19] and [20]. [21] proposes a scheme based on two online ensembles, one used for system predictions, the other one used to learn the new concept after a drift is detected. Weighted majority [22] is an ensemble techniques that maintain a collection of given learners, predict using a weighted majority rule, and decreases in a multiplicative manner

the weights of the learners in the pool that disagree with the label whenever the ensemble makes a mistakes. In [23] the weights of the learners that agree with the label when the ensemble makes a mistakes are increased, and the weights of the learners that disagree with the label are decreased also when the ensemble predicts correctly. To prevent the weights of the learners which performed poorly in the past from becoming too small with respect to the other learners, [24] proposes a modified version of weighted majority adding a phase, after the multiplicative weight update, in which each learner shares a portion of its weight with the other learners.

While many of the ensemble learning techniques have been developed assuming no a priori knowledge about the statistical properties of the data – as is required in most of the SPAs – these techniques are often designed for a centralized scenario. In fact, the base classifiers in these approaches are not distributed entities, they all observe the same data streams, and the focus of ensemble construction is on the statistical advantages of learning with an ensemble, with little study of learning under communication constraints. It is possible to cast these techniques within the framework of distributed learning, but as is they would suffer from many drawbacks. For example, [18]–[21] would require an entity that collects and stores all the data recently observed by the learners and that tells the learners how to adapt their local classifiers, which is clearly impractical in SPAs that need to process real-time streams characterized by high data rates.

Diffusion adaptation literature [25]–[32] consists of learning agents that are linked together through a network topology in a distributed setting. The agents must estimate some parameters based on their local observations and on the continuous sharing and diffusion of information across the network, and there is a focus on learning in distributed environments under communication constraints. In fact, [32] shows that a classification problem can be cast within the diffusion adaptation framework. However, there are some major constraints that are posed on the learners. First, in [25]–[32] all the learners are required to estimate the same set of parameters (i.e., they pursue a common goal) and combine their local estimates to converge toward a unique and optimal solution. This is a strong assumption for SPAs, as the learners might have different objectives and may use different information depending on what they observe and on their spatio-temporal position in the network. Hence, the optimal aggregation function may need to be specific to each learner.

There has been a large amount of recent work on building frameworks for distributed online learning with dynamic data streams, limited communication, delayed labels and feedback and self-interested and cooperative learners [33]–[36]. We discuss this briefly next.

To mine the correlated, high-dimensional and dynamic data instances captured by one or multiple heterogeneous data sources, extract actionable intelligence from these instances and make decisions in real-time as discussed previously, a few important questions need to be answered: Which processing/prediction/decision rule should a local learner (LL) select? How should the LLs adapt and learn their rules to maximize their performance? How should the processing/predictions/decisions of the LLs be combined/fused by a meta-learner to maximize the overall performance? Most literature treats the LLs as black box algorithms, and proposes various fusion algorithms for the ensemble learner with the goal of issuing predictions that are at least as good as the best LL in terms of prediction accuracy and the performance bounds proved for the ensemble in these works depend on the performance of the LLs. In [35] a the authors go one step further and study the joint design of learning algorithms for both the LLs and the ensemble. They present a novel systematic learning method (Hedge Bandits) which continuously learns and adapts the parameters of both the LLs and the ensemble, after each data instance, and provide both long run (asymptotic) and short run (rate of learning) performance guarantees. Hedge Bandits consists of a novel contextual bandit algorithm for the LLs and Hedge algorithm for the ensemble, and is able to exploit the adversarial regret guarantees of Hedge and the data-dependent regret guarantees of the contextual bandit algorithm to derive a data-dependent regret bound for the ensemble.

In [34], the ensemble learning consists of multiple distributed local learners, which analyze different streams of data correlated to a common classification event, and local predictions are collected and combined using a weighted majority rule. A novel online ensemble learning algorithm is then proposed to update the aggregation rule in order to adapt to the underlying data dynamics. This overcomes several limitations of prior work by allowing for a) *different* correlated data streams with statistical dependency among the label and the observation being different across learners, b) data being processed incrementally, once on arrival leading to improved scalability, c) support for different types of local classifiers including support vector machine, decision tree, neural networks, offline/online classifiers, etc., and d) allowing for asynchronous delays between the label arrival across the different learners. A modified version of this framework was applied to the problem of collision detection by networked sensors similarly to the one which we discussed on Section II of this chapter. For details, please refer to [37].

A more general framework, where the rule for making decisions and predictions is general, and depends on the costs and accuracy (specialization) of the autonomous learners was proposed in

[33]. This cooperative online learning scheme considers a) whether the learners can improve their detection accuracy by exchanging and aggregating information, b) whether the learners improve the timeliness of their detections by forming clusters, i.e., by collecting information only from surrounding learners, and c) whether, given a specific tradeoff between detection accuracy and detection delay, if it is desirable to aggregate a large amount of information, or is it better to focus on the most recent and relevant information.

In [38], these techniques are considered in a setting with a number of speed sensors, that are spatially distributed along a street and can communicate via an exogenously-determined network, and the problem of detecting in real-time collisions that occur within a certain distance from each sensor is studied.

In [36], a novel framework for decentralized, online learning by many self-interested learners is considered. In this framework, learners are modeled as cooperative contextual bandits, and each learner seeks to maximize the expected reward from its arrivals, which involves trading off the reward received from its own actions, the information learned from its own actions, the reward received from the actions requested of others and the cost paid for these actions - taking into account what it has learned about the value of assistance from each other learner. A distributed online learning algorithm is provided and analytic bounds to compare the efficiency of these with algorithms with the complete knowledge (oracle) benchmark (in which the expected reward of every action in every context is known by every learner) are established: regret - the loss incurred by the algorithm - is sublinear in time. These methods have been adapted in [39] to provide expertise discovery in medical environments. Here, an expert selection system is developed that learns online who is the best expert to treat a patient having a specific condition or characteristic.

In Subsection V-B we describe one such framework for online, distributed, ensemble learning that addresses some of the challenges discussed in Section III and is well suited for the transportation problem described in Section II. The presented methodology does not require a priori knowledge of the statistical properties of the data. This means that it can be applied both when a priori information is available and when a priori information is not available. However, if the statistical properties of the data are available beforehand, it may be convenient to apply schemes that are specifically designed to take into account the known statistical properties of the data. Moreover, the presented methodology does not require any specific assumption on the form of the loss or objective function. This means that any loss or objective function can be adopted. However, notice that the final performance of scheme depend on the selected function.

For illustrative purposes, in Subsection V-C we consider a specific loss function and we derive an adaptive algorithm based on this loss function, and in Subsection V-D we describe how the proposed framework can be adopted for a collision detection application.

B. Systematic Framework for Online Distributed Ensemble Learning

We now proceed to formalize the problem of large scale distributed learning from heterogeneous and dynamic data streams using the problem defined in Section II. Formally, we consider a set $\mathcal{K} = \{1, \ldots, K\}$ of *learners* that are geographically distributed and connected via *links* among pairs of learners. We say that there is a *link* (i, j) between learners i and j if they can communicate directly with each other. In the case of our congestion application, each learner observes part of the transportation network by consuming geographically local readings from sensors and phones within a region, and is linked to other learner streams via interfaces like the Export/Import interface described in Section IV.

Each learner is an ensemble of *local classifiers* that observes a specific set of data sources and relies on them to make local classifications, i.e., partition data items into multiple classes of interest[3]. In our application scenario, this maps to a binary classification task - predicting presence of congestion at a certain location within a certain time window. Each local classifier may be an arbitrary function (e.g., implemented using well known techniques such as neural networks, decision trees, etc.) that performs classification for the classes of interest. In Table reftbl:spl we show an implementation of this in a stream programming language with two local classifiers, `MySVM` and `MyNN` operators, and an aggregate `MyAggregation` operator. In order to simplify the discussion, we assume that each learner exploits a single local classifier and we focus on binary classification problems, but it is possible to generalize the approach to the multi-classifier and multi-class cases. Each learner is also characterized by a local *aggregation rule*, which is adaptive.

Raw data items in our application can include sensor readings from the transportation network and user phones, as well as information about the weather. These data items are cleaned, pre-processed, merged, and features are extracted from them (e.g., see Table II), which are then sent to the geographically appropriate learner. When a learner observes a feature vector, it first exploits the local classifier to make a local prediction, then it sends its local prediction to other learners in

[3]We present the ensemble learning in a classification setting, but the discussion is also applicable in a regression setting.

its neighborhood and receives local predictions from the other learners, before it finally exploits the aggregation rule to combine its local predictions and the predictions from its neighbors into a final classification.

For simplicity of exposition, we consider a discrete time model in which time is divided into *slots*, but an extension to a continuous time model is possible. At the beginning of the n-th time slot, K multi-dimensional *instances* $\mathbf{x}_i^n \in \mathcal{X}_i$, $i = 1 \dots K$, and K *labels* $y_i^n \in \{-1, +1\}$, $i = 1 \dots K$, are drawn from an underlying and unknown joint probability distribution.. Each learner i observes the instance $\mathbf{x}_i^n$ and its task is to predict the label y_i^n – see Fig. 6. We shall assume that a label y_j^n is correlated with all the instances $\mathbf{x}_i^n$, $i = 1 \dots K$. In this way, learner i's observation can contain information about the label that learner j has to predict. We remark that this correlation is not known beforehand.

Each learner i is equipped with a *local classifier* $f_i^n : \mathcal{X}_i \to \{-1, +1\}$ that generates the *local prediction* $s_i^n \triangleq f_i^n(\mathbf{x}_i^n)$ based on the observed instance $\mathbf{x}_i^n$ at time slot n. Our framework can accommodate both static pre-trained classifiers, and adaptive classifiers that learn online the parameters and configurations to adopt [40]. However, the focus of this section will not be on classifier design, for which many solutions already exist (e.g., support vector machines, decision trees, neural networks, etc.); instead, we will focus on how the learners exchange and learn how to aggregate the local predictions generated by the classifiers.

We allow the distributed learners to exchange and aggregate their local predictions through multi-hop communications; however, within one time slot a learner can send only a single transmission to each of its neighbors. We denote by $\overline{s}_{ij}^n$ learner j's local prediction possessed by learner i before the aggregation at time instant n. The information is disseminated in the network as follows. First, each learner i observes $\mathbf{x}_i^n$ and updates $\overline{s}_{ii}^n = s_i^n = f_i^n(\mathbf{x}_i^n)$. Next, learner i transmits to each neighbor j the local prediction s_i^n and the local predictions $\overline{s}_{ik}^{n-1}$, for each learner $k \neq i$ such that the link (i, j) belongs to the shortest path between k and j. Hence, if transmissions are always correctly received we have $\overline{s}_{ii}^n = s_i^n$ and $\overline{s}_{ij}^n = s_j^{n-d_{ij}+1}$, $i \neq j$, where d_{ij} is the distance in number of hops between i and j. For instance, Fig. 5 represents the flow of information toward learner 1 for a binary tree network assuming that transmissions are always correctly received. More generally, if transmissions can be affected by communication errors we have $\overline{s}_{ij}^n = s_j^m$, for some $m \leq n - d_{ij} + 1$.

Each learner i employs a *weighted majority aggregation* rule to fuse the data it possesses, and

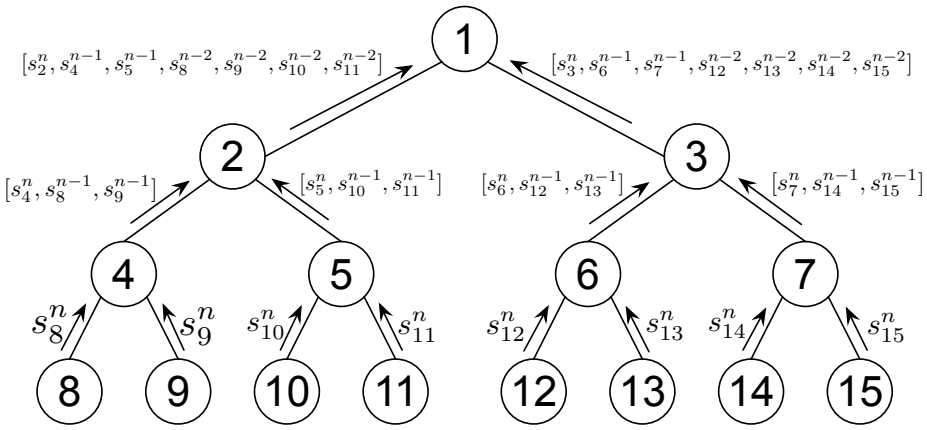

Fig. 5: Flow of information toward learner 1 at time slot n for a binary tree network.

generates a final prediction $\hat{y}_i^n$ as follows:

$$\hat{y}_i^n \triangleq \mathrm{sgn}\left(\sum_{j\in\mathcal{K}} w_{ij}^n \overline{s}_{ij}^n\right) \triangleq \begin{cases} +1, & \text{if argument of sgn is } \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{1}$$

where $\mathrm{sgn}(\cdot)$ is the sign function.

In the above construction, learner i first aggregates all possessed predictions $\{\overline{s}_{ij}^n\}$ using the weights $\{w_{ij}^n\}$, and then uses the sign of the fused information to output its final classification, $\hat{y}_i^n$. While weighted majority aggregation rules has been considered before in the ensemble learning literature [17]–[20], there is an important distinction in Eq. (1) that is particularly relevant to the online distributed stream-mining context: since we are limiting the learners to exchange information only via links, learners receive information from other learners with delay (i.e., in general $\overline{s}_{ij}^n \neq s_j^n$), as a consequence different learners have different information to exploit (i.e., in general $\overline{s}_{ij}^n \neq \overline{s}_{kj}^n$).

Each learner i maintains a total of K weights and K local predictions, which we collect into vectors:

$$\mathbf{w}_i^n \triangleq (w_{i1}^n, \ldots, w_{iK}^n) \quad ; \quad \mathbf{s}_i^n \triangleq (\overline{s}_{i1}^n \ldots, \overline{s}_{iK}^n) \tag{2}$$

Given the weight vector $\mathbf{w}_i^n$, the decision rule (1) allows for a geometric interpretation: the homogeneous hyperplane in $\Re^K$ which is orthogonal to $\mathbf{w}_i^n$ separates the positive prediction (i.e., $\hat{y}_i^n = +1$) form the negative predictions (i.e., $\hat{y}_i^n = -1$).

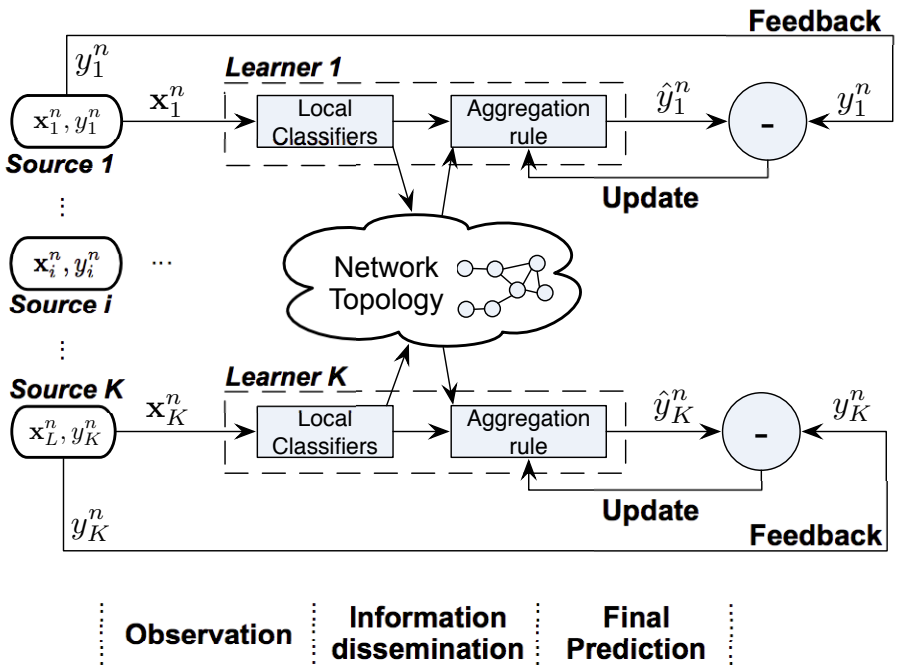

Fig. 6: System model described in Subsection V-B.

We consider an *online learning setting* in which the true label y_i^n is eventually observed by learner i. Learner i can then compare both $\hat{y}_i^n$ and y_i^n and use this information to update the weights it assigns to the other learners. Indeed, since we do not assume any a priori information about the statistical properties of the processes that generate the data observed by the various learners, we can only exploit the available observations, and the history of past data, to guide the design of the adaptation process over the network.

In summary, the sequence of events that takes place in a generic time slot n, represented in Fig. 6, involves five phases:

1) *Observation:* Each learner i observes an instance $\mathbf{x}_i^n$ at time n.

2) *Information Dissemination:* Learners send the local predictions they possess to their neighbors.

3) *Final Prediction:* Each learner i computes and outputs its final prediction $\hat{y}_i^n$.

4) *Feedback:* Learners can observe the true label y_i^m that refers to a time slot $m \leq n$.

5) *Adaptation:* If y_i^m is observed, learner i updates its aggregation vector from $\mathbf{w}_i^n$ to $\mathbf{w}_i^{n+1}$.

In the context of the discussed framework, it is fundamental to develop strategies for adapting the aggregation weights $\{w_{ji}^n\}$ over time, in response to how well the learners perform. A possible

approach is discussed next.

C. Online Learning of the Aggregation Weights

A possible approach to update the aggregation weights is to associate with each learner i an instantaneous loss function $\ell_i^n(\mathbf{w}_i)$ and minimize, with respect to the weight vector $\mathbf{w}_i$, the cumulative loss given all observations up to time slot n. In the following we consider this approach, adopting an instantaneous *hinge loss function* [41]:

For each time instant n we consider the one-shot *loss function*

$$\ell_i^n(\mathbf{w}_i) \triangleq \begin{cases} -\alpha^{MD}\mathbf{w}_i \cdot \mathbf{s}_i^n & \text{if } \hat{y}_i^n = 0 \text{ and } y_i^n = 1 \\ \alpha^{FA}\mathbf{w}_i \cdot \mathbf{s}_i^n & \text{if } \hat{y}_i^n = 1 \text{ and } y_i^n = 0 \\ 0 & \text{if } \hat{y}_i^n = y_i^n \end{cases} \tag{3}$$

where the parameters $\alpha^{MD} > 0$ and $\alpha^{FA} > 0$ are the *mis-detection* and *false alarm* unit costs, and $\mathbf{w}_i \cdot \mathbf{s}_i^n \triangleq \sum_{j \in \mathcal{K}} w_{ji} \overline{s}_{ij}^n$ denotes the scalar product between $\mathbf{w}_i$ and $\mathbf{s}_i^n$. The hinge loss function is equal to 0 if the weight vector $\mathbf{w}_i$ allows to predict correctly the label y_i^n, otherwise the value of the loss function is proportional to the distance of $\mathbf{s}_i^n$ from the separating hyperplane defined by $\mathbf{w}_i$, multiplied by α^{MD} if the prediction is 0 but the label is 1 – we refer to this type of error as a *mis-detection* – or multiplied by α^{FA} if the prediction is 1 but the label is 0 – we refer to this type of error as a *false alarm*.

The hinge loss function gives higher importance to errors that are more difficult to correct with the current weight vector. A related albeit different approach is adopted in Adaboost [17], in which the importance of the errors increase exponentially in the distance of the local prediction vector from the separating hyperplane. Here, however, the formulation is more general and allows for the diffusion of information across neighborhoods simultaneously, as opposed to assuming each learner has access to information from across the entire set of learners in the network.

We can then formulate a global objective for the distributed stream mining problem as that of determining the optimal weights by minimizing the cumulative loss given all observations up to time slot n:

$$\left\{\mathbf{w}_i^{n+1}\right\}_{i=1}^K = \underset{\{\mathbf{w}_i\}_{i=1}^K}{\operatorname{argmin}} \sum_{i=1}^K \sum_{m=1}^n \overline{\ell}_i^m \tag{4}$$

where $\overline{\ell}_i^m \triangleq \ell_i^m$ if y_i^m has been observed by time instant n, otherwise $\overline{\ell}_i^m \triangleq 0$.

To solve (4) learner i must store all previous labels and all previous local predictions of all the learners in the system, which is impractical in SPAs, where the volume of the incoming

data is high and the number of learners is large. Hence, we adopt the stochastic gradient descent algorithm to incrementally approach the solution of (4) using only the most recent observed label. If label y_i^m is observed at the end of time instant n, we obtain the following update rule for $\mathbf{w}_i^n$:

$$\mathbf{w}_i^{n+1} = \begin{cases} \mathbf{w}_i^n + \alpha^{MD}\mathbf{s}_i^n & \text{if } \tilde{y}_i^m = 0 \text{ and } y_i^m = 1 \\ \mathbf{w}_i^n - \alpha^{FA}\mathbf{s}_i^n & \text{if } \tilde{y}_i^m = 1 \text{ and } y_i^m = 0 \\ \mathbf{w}_i^n & \text{if } \tilde{y}_i^n = y_i^n \end{cases} \tag{5}$$

where $\tilde{y}_i^m$ is the prediction that learner i would have made at time instant m with the current weight vector $\mathbf{w}_i^n$. This construction allows a meaningful interpretation. It shows that learner i should maintain its level of confidence in its data when its decision agrees with the observed label. If disagreement occurs, then learner i needs to assess which local predictions lead to the mis-classification: the weight w_{ij}^n that learner i adopts to scale the local predictions it receives from learner j is increased (by either α^{MD} or α^{FA} units, depending on the type of error) if the local prediction sent by j agreed with the label, otherwise w_{ij}^n is decreased.

[7] and [8] derive worst-case upper bounds for the mis-classification probability of a learner adopting the update rule 5. Such bounds are expressed in terms of the mis-classification probabilities of two benchmarks: 1) the mis-classification probability of the best local classifiers and 2) the mis-classification probability of the best linear aggregator. We remark that the best local classifiers and the best linear aggregator are not known and cannot be computed beforehand; in fact, this would require to know in advance the realization of the process that generates the instances and the labels.

The optimization problem (4) can also be solved within the diffusion adaptation framework, as proposed by [32]. In this framework the learners combine information with their neighbors, for example in the Combine Then Adapt (CTA) diffusion scheme they first combine their weights and then adopt the stochastic gradient descent [32]. Fig 7 illustrates the difference between our approach and the CTA scheme.

We remark that the framework described so far requires each learner to maintain a weight (i.e., an integer value) and a local prediction (i.e., a Boolean value) for each other learner in the network. This means that the memory and computational requirements scale linearly in the number of learners K. However, notice that the aggregation rule 1 and the update rule 5 only require basic operations such as add, multiply, and compare. Moreover, if the learners have a common goals, i.e., they must predict a common class label y^n, it is possible to develop a scheme in which each learner keeps track only of its own local prediction and of the weight

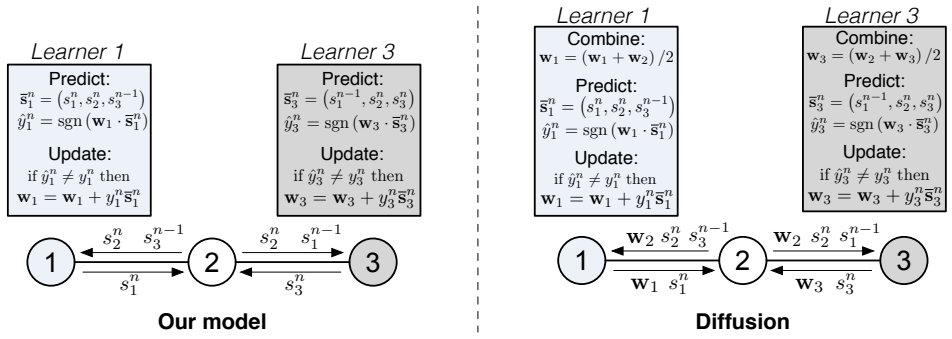

Fig. 7: A comparison between the proposed algorithm and the Combine Then Adapt (CTA) scheme in terms of information dissemination and weight update rule.

used to scale its local prediction, and is responsible to update such a weight. In this scheme the learners exchange the weighted local predictions instead of the local predictions and the memory and computational requirements scale as a constant in the number of learners K. For additional details, we refer the reviewer to [8].

The framework discussed in this subsection naturally maps onto a deployment using a stream processing system. Each of the learners shown in Fig. 6 map onto the subgraph labeled Learner in Fig. 2, with the local classifiers mapping onto the shown parallel topology, and the aggregation rule mapping to the fan-in on that subgraph. As mentioned earlier, the base classifiers may be implemented using the toolkits provided by systems like Streams that include wrappers for R and Matlab. The feedback y_i^n corresponds to the Delayed Feedback in Fig. 2, and the input feature vector $\mathbf{x}_i^n$ is computed by the Pre-proc part of the subgraph in Fig. 2, and can include different types of spatio-temporal processing and feature extraction. Finally, the communication between the learners in Fig. 6 is enabled by the Learner Information Exchange connections in Fig. 2. In summary, this online, distributed, ensemble learning framework can naturally be implemented on a stream processing platform. This combination is very powerful, as it now allows the design, development, and deployment of such large-scale complex applications much more feasible, and it also enables a range of novel signal processing, optimization, and scheduling research. We discuss some of these open problems in Section VI.

D. Collision Detection Application

In this Subsection we apply the framework described in Subsections V-B and V-C to a collision detection application in which a set of speed sensors – that are spatially distributed along a street

Algorithm Aggregation and update rules described in Subsections V-B and V-C

Initialization: $w_{ij}^n = 0$, $\forall\, i, j \in \mathcal{K}$

For each learner i and time instant n

  Observe $\mathbf{x}_i^n$ and update $\overline{s}_{ii}^n \leftarrow f_i^n(\mathbf{x}_i^n)$

  Send $\overline{s}_{ij}^n$ to all the neighbors k such that i is in the path between j and k, $\forall\, j$

  Update $\overline{s}_{ij}^n \leftarrow \overline{s}_{kj}^n$ for each k and j such that $\overline{s}_{kj}^n$ is correctly received

  Predict $\hat{y}_i^n \leftarrow \text{sgn}\left(\sum_{j\in\mathcal{K}} w_{ij}^n \overline{s}_{ij}^n\right)$

  For each time instant m such that y_i^m is observed

    If $\text{sgn}\left(\sum_{j\in\mathcal{K}} w_{ij}^n \overline{s}_{ij}^m\right) \neq y_i^m$

      If $y_i^m = 1$ **do** $\mathbf{w}_i^n \leftarrow \mathbf{w}_i^n + \alpha^{MD}\mathbf{s}_i^n$

      Else do $\mathbf{w}_i^n \leftarrow \mathbf{w}_i^n - \alpha^{FA}\mathbf{s}_i^n$

– must detect in real-time collisions that occur within a certain distance from them.

We consider a set $\mathcal{K} = \{1, \ldots, K\}$ of K speed sensors that are distributed along both travel directions of a street – see the left side of Fig. 8. We focus on a generic sensor i that must detect the occurrence of *collision events* within z miles from its location along the corresponding travel direction, where z is a predetermined parameter. A collision event e_ℓ is characterized by an unknown *starting time* $t_{\ell,start}$ – when the collision occurs – and an unknown ending time $t_{\ell,end}$ – when the collision is cleared. The goal of sensor i is to detect the collision e_ℓ by the time $t_{\ell,det} = t_{\ell,start} + T_{max}$, where T_{max} can be interpreted as the maximum time after the occurrence of the collision such that the information about the collision occurrence can be exploited to take better informed actions (e.g., the average time after which a collision is reported by other sources).

We divide the time into *slots* of length T. We write $y_i^n = +1$ if a collision occurs at or before time instant n and is not cleared by time instant n, whereas we write $y_i^n = -1$ to represent the absence of a collision. Fig. 9 illustrates these notations.

At the beginning of the n-th time slot each speed sensor j observes a speed value $x_j^n \in \Re$, which represents the average speed value of the cars that have passed through sensor j from the beginning of the $(n-1)$-th time slot until the beginning of the n-th time slot. We consider a threshold-based classifier:

$$s_j^n = f_j^n(x_j^n) \triangleq \begin{cases} -1 & \text{if } x_j^n > \beta_j v_j^n \\ +1 & \text{if } x_j^n \leq \beta_j v_j^n \end{cases} \tag{6}$$

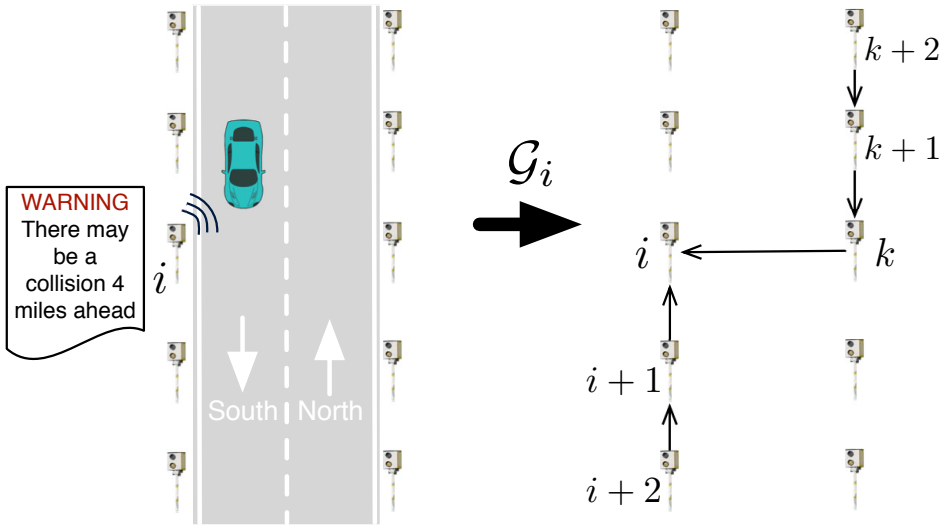

Fig. 8: A generic speed sensor i must detect collisions in real-time and inform the drivers (left). To achieve this goal sensor i receives the observations from the other sensors, the flow of information is represented by a directed graph $\mathcal{G}_i$ (right).

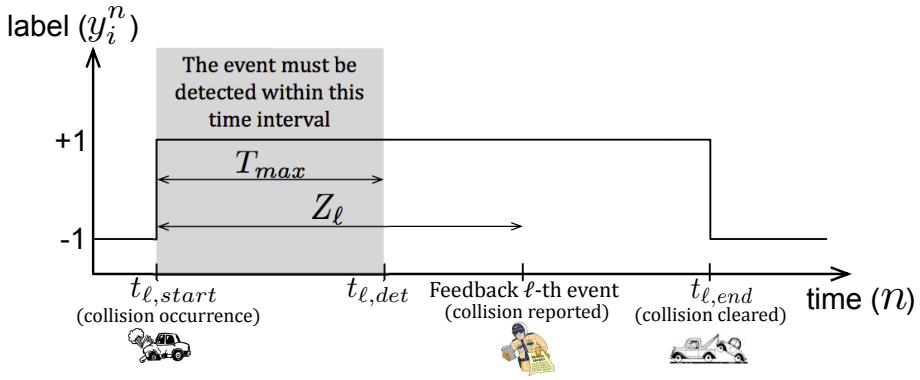

Fig. 9: Illustration of the considered notations.

where v_j^n is the average speed observed by sensor j during that day of the week and time of the day, and $\beta_j \in [0,1]$ is a threshold parameter.

If sensor j is close to sensor i the speed value x_j^n and the local prediction s_j^n are correlated to the occurrence or absence of the collision events that sensor i must detect. For this reason, to detect collisions in an accurate and timely manner, the sensors must exchange their local predictions. Specifically, we denote the sensors such that sensor i precedes sensor $i+1$ in the

direction of travel. In order to detect whether a collision has occurred within z miles from its location, sensor i requires the observations of the subsequent sensors in the direction of travel (e.g., sensors $i+1$, $i+2$, etc.), up to the sensor that is far from sensor i more than z miles. Hence, the information flows in the opposite direction with respect to the direction of travel. Such a scenario is represented by the right side of Fig. 8. Notice that sensor $i+1$ is responsible to collect the observation from sensor $i+2$ and to send to sensor i both its observation and the observation of $i+2$.

Fig. 8 shows also the flow of information provided by one side of the street to the other side of the street (i.e., from sensor k to sensor i). Indeed, the fact that the observations on one side of the street are not influenced by a collision on the other side of the street can be extremely useful to assess the traffic situation and distinguish between collisions and other types of incidents. For example, the sudden decrease of the speed observed by some sensors in the considered travel direction may be a collision warning sign; however, if at the same time instants the speed observed by the sensors in the opposite travel direction decreases as well, then an incident that affect both travel directions may have occurred (e.g., it started to rain) instead of a collision.

We can formally define the flow of information represented by the right side of Fig. 8 with a directed graph $\mathcal{G}_i \triangleq (\mathcal{K}_i, \mathcal{L}_i)$,[4] where $\mathcal{K}_i \subset \mathcal{K}$ is the subset of sensors that send their local predictions to sensor i (included i itself), and $\mathcal{L}_i \subset \mathcal{K}_i \times \mathcal{K}_i$ is the set of links among pairs of sensors.

Now both the local classifiers and the flow of information are defined, learner i can adopt the framework described in Subsections V-B and V-C to detect the occurrence of collisions within z miles from its location. Specifically, learner i maintains in memory K_i weights (K_i is the cardinality of $\mathcal{K}_i$) that are collected in the weight vector $\mathbf{w}_i^n$, it predicts adopting 1, and it updates the weights adopting 5 whenever a feedback is received. The feedback about the occurrence of the collision event e_ℓ can be provided, for example, by a driver or by a police officer, and it is in general received with delay. In Fig. 8 such a delay is denoted by Z_ℓ.

We have evaluated the proposed framework over real word datasets. Specifically, we have exploited a dataset containing the speed readings of the loop sensors that are distributed along

[4]We remark that we focus on a particular sensor i to keep the discussion and the notations simple. However, all the sensors may be required to detect collisions that are within z miles from their location. Hence, each sensor applies the same scheme we propose in this paper for a generic sensor i. This means that there are many directed graphs (e.g., $\mathcal{G}_1$, $\mathcal{G}_2$, etc.) representing how information flows among different sensors.

a 50 mile segment of the freeway 405 that passes through Los Angeles County, and a collision dataset containing the reported collisions that occurred along the freeway 405 during the months of September, October, and November 2013. For a more detailed description of the datasets we refer the reader to [42]. Our illustrative results show that the considered framework is able to detect more than half of the collisions occurring within a distance of 4 miles from a specific sensors, while generating false alarms in the order of 1 false alarm every 100 predictions. The results show also that by setting the ratio $\frac{\alpha^{MD}}{\alpha^{FA}}$ mis-detections and false alarms can be traded-off.

VI. What Lies Ahead

There are several open research problems at this application-algorithm-systems interface – needed for fog computing – that are worth investigating. First, there is currently no principled approach to decompose an online distributed large-scale learning problem into a topology/flow-graph of streaming operators and functions. While standard engineering principles of modularity, reuse, atomicity, etc. apply, there is no formalism that supports such a decomposition.

Second, there are several optimization problems related to mapping a given processing topology onto physical processes that can be instantiated on a distributed computation platform. This requires a multi-objective optimization where communication costs need to be traded off with memory and computational costs, while ensuring efficient utilization of resources. Also, given that resource requirements and data characteristics change over time, these optimization problems may need to be solved incrementally or periodically. The interaction between these optimizations and the core learning problem needs to be also formally investigated.

Third, there are several interesting topology configuration and adaptation problems that can be considered: learners can be dynamically switched on or off to reduce system resource usage or improve system performance, the topology through which they are connected can adapt to increase parallelism, the selectivity / operating points of individual classifiers can be modified to reduce workloads on downstream operators, past data and observations can be dynamically dropped to free memory resources, etc. The impact of each individual adaptation and of the interaction among different levels of adaptation is unclear and needs to be investigated. Some examples of exploiting these tradeoffs have been considered in [11], but this is a fertile space for future research.

Another important extension is the use of active learning approaches [43] to gather feedback in cases where it is sparse, hard, or costly to acquire.

Fourth, there is need to extend the meta-learning aggregation rule from a linear form to other forms (e.g., decision trees) to exploit the decision space more effectively. Additionally, meta-learners may themselves be hierarchically layered into multiple levels – with different implications for learning, computational complexity and convergence.

Fifth, in the presence of multiple learners, potentially belonging to different entities, these ensemble approaches need to handle non-cooperative, and in some cases even malicious entities. In [44], a few steps have been taken in this direction. This work studies distributed online recommender systems, in which multiple learners, that are self-interested and represent different companies/products are competing and cooperating to jointly recommend products to users based on their search query as well as their specific background including history of bought items, gender and age.

Finally, while we have posed the problem of distributed learning in a supervised setting (i.e., the labels are eventually observed), there is also need to build large-scale online algorithms for knowledge discovery in semi-supervised and unsupervised settings. Constructing online ensemble methods for clustering, outlier detection, and frequent pattern mining are interesting directions. A few steps in this directions have been taken in [45] and [46], where context-based unsupervised ensemble learning was proposed and clustering, respectively.

More discussion of such complex applications built on a stream processing platform, open research problems, and a more detailed literature survey may be obtained from [9]. Overall, we believe that the space of distributed, online, large-scale, ensemble learning using stream processing middleware is an extremely fertile space for novel research and construction of real-world deployments that have the potential to accelerate our effective use of streaming Big Data to realize a Smarter Planet.

REFERENCES

[1] "IBM Smarter Planet," http://www.ibm.com/smarterplanet, retrieved October, 2012.

[2] "ITU Report: Measuring the Information Society," http://www.itu.int/dms_pub/itu-d/opb/ind/D-IND-ICTOI-2012-SUM-PDF-E.pdf, 2012.

[3] "IBM InfoSphere Streams," www.ibm.com/software/products/en/infosphere-streams/, retrieved March, 2011.

[4] "Storm Project," http://storm-project.net/, retrieved October, 2012.

[5] "Apache Spark," http://spark.apache.org/, retrieved September, 2015.

[6] Y. Zhang, D. Sow, D. S. Turaga, and M. van der Schaar, "A fast online learning algorithm for distributed mining of bigdata," in *The Big Data Analytics Workshop at SIGMETRICS*, 2013.

[7] L. Canzian, Y. Zhang, and M. van der Schaar, "Ensemble of distributed learners for online classification of dynamic data streams," UCLA, Electrical Engineering Department, Tech. Rep., 2013. [Online]. Available: http://arxiv.org/abs/1308.5281

[8] L. Canzian and M. van der Schaar, "A network of cooperative learners for data–driven stream mining," *accepted and to appear in IEEE ICASSP*, 2014.

[9] H. Andrade, B. Gedik, and D. Turaga, *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. Cambridge University Press, 2014.

[10] C. Aggarwal, Ed., *Data Streams: Models and Algorithms*, 2007.

[11] R. Ducasse, D. S. Turaga, and M. van der Schaar, "Adaptive topologic optimization for large-scale stream mining," *IEEE J. Sel. Topics Signal Process.*, vol. 4, no. 3, pp. 620–636, 2010.

[12] S. Ren and M. van der Schaar, "Efficient resource provisioning and rate selection for stream mining in a community cloud," *IEEE Trans. Multimedia*, vol. 15, no. 4, pp. 723–734, 2013.

[13] F. Fu, D. S. Turaga, O. Verscheure, M. van der Schaar, and L. Amini, "Configuring competing classifier chains in distributed stream mining systems," *IEEE J. Sel. Areas Commun.*, vol. 1, no. 4, pp. 548–563, 2007.

[14] A. Beygelzimer, A. Riabov *et al.*, "Big data exploration via automated orchestration of analytic workflows," *USENIX International Conference on Automated Computing*, 2013.

[15] Z. Haipeng, S. R. Kulkarni, and H. V. Poor, "Attribute-distributed learning: Models, limits, and algorithms," *IEEE Trans. Signal Process.*, vol. 59, no. 1, pp. 386–398, 2011.

[16] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[17] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, Aug. 1997.

[18] W. Fan, S. J. Stolfo, and J. Zhang, "The application of AdaBoost for distributed, scalable and on-line learning," in *Proc. ACM SIGKDD*, 1999, pp. 362–366.

[19] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. ACM SIGKDD*, 2003, pp. 226–235.

[20] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Integrating novel class detection with classification for concept-drifting data streams," in *Proc. ECML PKDD*, 2009, pp. 79–94.

[21] L. L. Minku and Y. Xin, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 4, pp. 619–633, 2012.

[22] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Inf. Comput.*, vol. 108, no. 2, pp. 212–261, Feb. 1994.

[23] A. Blum, "Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain," *Mach. Learn.*, vol. 26, no. 1, pp. 5–23, Jan. 1997.

[24] M. Herbster and M. K. Warmuth, "Tracking the best expert," *Mach. Learn.*, vol. 32, no. 2, pp. 151–178, Aug. 1998.

[25] A. Ribeiro and G. B. Giannakis, "Bandwidth-constrained distributed estimation for wireless sensor networks-part i: Gaussian case," *IEEE Trans. Signal Process.*, vol. 54, no. 3, pp. 1131–1143, 2006.

[26] ——, "Bandwidth-constrained distributed estimation for wireless sensor networks-part ii: unknown probability density function," *IEEE Trans. Signal Process.*, vol. 54, no. 7, pp. 2784–2796, 2006.

[27] J.-J. Xiao, A. Ribeiro, Z.-Q. Luo, and G. B. Giannakis, "Distributed compression-estimation using wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 23, no. 4, pp. 27–41, 2006.

[28] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "Distributed learning for decentralized inference in wireless sensor networks," *IEEE Trans. Signal Process.*, vol. 23, no. 4, pp. 56–69, 2006.

[29] H. Zhang, J. Moura, and B. Krogh, "Dynamic field estimation using wireless sensor networks: Tradeoffs between estimation error and communication cost," *IEEE Trans. Signal Process.*, vol. 57, no. 6, pp. 2383–2395, 2009.

[30] S. Barbarossa and G. Scutari, "Decentralized maximum-likelihood estimation for sensor networks composed of nonlinearly coupled dynamical systems," *IEEE Trans. Signal Process.*, vol. 55, no. 7, pp. 3456–3470, 2007.

[31] A. Sayed, S.-Y. Tu, J. Chen, X. Zhao, and Z. Towfic, "Diffusion strategies for adaptation and learning over networks: an examination of distributed strategies and network behavior," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 155–171, 2013.

[32] Z. J. Towfic, J. Chen, and A. H. Sayed, "On distributed online classification in the midst of concept drifts," *Neurocomputing*, vol. 112, pp. 139–152, 2013.

[33] L. Canzian and M. van der Schaar, "Timely event detection by networked learners," *IEEE Transactions on Signal Processing*, vol. 63, no. 5, pp. 1282–96, 2015.

[34] L. Canzian, Y. Zhang, and M. van der Schaar, "Ensemble of distributed learners for online classification of dynamic data streams," *IEEE Transactions on Signal ans Information Processing over Networks*, 2015.

[35] C. Tekin and M. van der Schaar, "Active learning in context-driven stream mining with an application to image mining," *IEEE Transactions Image Processing*, 2015.

[36] ——, "Distributed online learning via cooperative contextual bandits," *IEEE Transactions Signal Processing*, vol. 63, no. 14, pp. 3700–14, 2015.

[37] L. Canzian and M. van der Schaar, "Collision detection by networked sensors," *Transactions on Signal and Information Processing in Networks*, 2015.

[38] J. Xu, D. Deng, U. Demiryurek, C. Shahabi, and M. van der Schaar, "Mining the situation: Spatiotemporal traffic prediction with big data," *IEEE Journal on Selected Topics in Signal Processing*, 2015.

[39] L. Canzian and M. van der Schaar, "Collision detection by networked sensors," *Transactions on Signal and Information Processing in Networks*, 2015.

[40] D. Shutin, S. R. Kulkarni, and H. V. Poor, "Incremental reformulated automatic relevance determination," *IEEE Trans. Signal Process.*, vol. 60, no. 9, pp. 4977–4981, 2012.

[41] L. Rosasco, E. D. Vito, A. Caponnetto, M. Piana, and A. Verri, "Are loss functions all the same?" *Neural Comput.*, vol. 16, no. 5, pp. 1063–1076, May 2004.

[42] B. Pan, U. Demiryurek, C. Gupta, and C. Shahabi, "Forecasting spatiotemporal impact of traffic incidents on road networks," in *IEEE ICDM*, 2013.

[43] M.-F. Balcan, S. Hanneke, and J. W. Vaughan, "The true sample complexity of active learning," *Machine learning*, vol. 80, no. 2-3, pp. 111–139, 2010.

[44] C. Tekin, S. Zhang, and M. van der Schaar, "Distributed online learning in social recommender systems," *IEEE Journal of Selected Topics in Signal Processing*, vol. 8, no. 4, pp. 638–52, 2014.

[45] E. Soltanmohammadi, M. Naraghi-Pour, and M. van der Schaar, "Context-based unsupervised data fusion for decision making," *International Conference on Machine Learning (ICML)*, 2015.

[46] D. Katselis, C. Beck, and M. van der Schaar, "Ensemble online clustering through decentralized observations," *CDC*, 2014.