Effective Use Methods for Continuous Sensor Data Streams in Manufacturing Quality Control

William Andrew Hitchcock

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2023

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial
fulfillment of the requirements for graduation from the
Honors Program of Liberty University.

_____

Frank Tuzi, Ph.D.
Thesis Chair

_____

Michael Zamperini, M.S.
Committee Member

_____

Emily C. Knowles, D.B.A.
Assistant Honors Director

_____

Date

Abstract

This work outlines an approach for managing sensor data streams of continuous numerical data in product manufacturing settings, emphasizing statistical process control, low computational and memory overhead, and saving information necessary to reduce the impact of nonconformance to quality specifications. While there is extensive literature, knowledge, and documentation about standard data sources and databases, the high volume and velocity of sensor data streams often makes traditional analysis unfeasible. To that end, an overview of data stream fundamentals is essential. An analysis of commonly used stream preprocessing and load shedding methods follows, succeeded by a discussion of aggregation procedures. Stream storage and querying systems are the next topics. Further, existing machine learning techniques for data streams are presented, with a focus on regression. Finally, the work describes a novel methodology for managing sensor data streams in which data stream management systems save and record aggregate data from small time intervals, and individual measurements from the stream that are nonconforming. The aggregates shall be continually entered into control charts and regressed on. To conserve memory, old data shall be periodically reaggregated at higher levels to reduce memory consumption.

Keywords: Data streams, statistical process control, quality control, aggregation, control charts

Effective Use for Continuous Sensor Data Streams in Manufacturing Quality Control

As technology becomes more advanced, sensors are becoming more prevalent: they are indispensable in numerous use cases, especially those related to condition monitoring and object tracking (Geisler, 2013). Gaber (2005) asserted that both applications are common in manufacturing environments, as RFID is often employed to pinpoint material and product locations within facilities, and measurements are automatically collected for quality assurance. These frequent measurements form data streams which are uniquely difficult to process due to high volume, velocity, and volatility (Gaber, 2005). These three factors often necessitate unorthodox analysis techniques; traditional relational database architectures frequently fail to suffice, according to Olken (2008). Sensor data streams are characterized by large volume (numerous measurements) and appreciable velocity (the measurements arrive at a swift rate). Together, volume and velocity create the demand for tremendous, sometimes prohibitive memory requirements if all entries are to be stored and processed using a traditional relational database, especially because the length of the stream is typically indefinite. To derive meaningful insights, processing algorithms must be efficient enough to keep pace with new data as it arrives, instead of running on accumulated historical data (Olken, 2008). Finally, some sensor data streams in manufacturing can be volatile: in some situations, trends can change frequently, meaning that old numbers are of little value and analysis must be conducted in real time (Gaber, 2005).

This work responds to the distinctive challenges of processing continuous data streams in manufacturing by first providing an overview of data stream concepts and introducing common management system architectures. To that end, established data stream preprocessing and aggregation practices are detailed. An exposition of data stream management system processes

of storage and querying follows. The next topics of discussion are mining and very fast machine learning (VFML); regression is the primary focus, as it is the most common VFML technique used on sensor data streams. This background culminates in the presentation of an original quality control methodology for high volume, high velocity sensor data streams measuring continuous physical data in a manufacturing setting.

Data Streams and Management Systems

Olken (2008) asserted that continuous, endless arrival of new information from sensors challenges constraints on resources used to store and query data streams and presents needs that traditional database management systems (DBMS) cannot meet. Memory, although storage methods are constantly advancing, is still finite: every value in an infinite stream with high volume and velocity cannot be stored forever (Olken, 2008). Geisler (2013) explained that systems must practice load shedding, the systematic selective elimination of data tuples. Load shedding methods include storing only averages of points over specific time intervals, sampling points from a time interval to store, or using Markovian models (Geisler, 2013). Sensor energy requirements and communication bandwidth restrictions are also essential to consider (Olken, 2008). Computational requirements are also limited; in many scenarios, the algorithms for summarizing the data and deriving insights can only a single pass over incoming data values, or else they will not be able to keep up with the stream (Read, 2020).

Geisler (2013) posited that data stream management systems (DSMS) address the unique difficulties associated with summarizing and processing data streams, leveraging numerous components to operate on inputted data points. Raw data is first sent through an input manager which takes source tuples, buffers them, and ensures proper order based on timestamp. The stream manager then finishes reformatting the raw data based on DBMS structure, which is often

very similar to a typical relational database (Geisler, 2013). Krempl (2014) said that, depending on the DBMS, the input manager, source manager, or another dedicated component will assume responsibility for preprocessing, which applies procedures to reconcile missing values. Preprocessing further entails removing noisy fluctuations, duplicate points, unwanted outliers, and missing values (Krempl, 2014).

Once the data are correctly configured and preprocessed, Geisler (2013) continued that a router component places the data in a queue for a query operator. A queue manager element is responsible for overseeing the operator queues and their respective buffers. It also directs data into secondary storage if RAM availability is insufficient. The queue manager collaborates with a storage manager, which brokers access to secondary storage when stream data is queried against old data or when stream data is archived. Flow of control subsequently passes to a scheduler that sequences the execution of various operators, and a query processor that acts on the stream. DBMSs use one of three methodologies for scheduling queries: time driven systems execute queries on given time intervals, tuple driven systems run queries as each new point arrives, and event driven architectures evaluate queries when specific trigger conditions are met. Dedicated load shedders enhance this main processing sequence, controlling data volumes by discarding tuples selected by sampling methods. The load shedders are folded into or supplemented by query optimizers, which dynamically change the DBMS's protocols based on diagnostic statistics (Geisler, 2013).

Individual DBMSs components leverage numerous, varied techniques to reduce data volumes and derive insights. Gaber (2005) grouped them into categories: data-based techniques entail summarizing a dataset or selecting a subset for analysis, task-based techniques leverage original or modified methods to address the unique difficulties of data stream processing, and

mining techniques find patterns in the stream. Common data-based techniques include stream sampling, load shedding, developing synopsis structures such as histograms and quantiles, and aggregation. Among task-based techniques are approximation algorithms to obtain solution estimates using stream data, sliding windows for memory management that shed or reduce old data as new information comes in, and algorithm output granularity to adapt procedures based on the availability of computational and storage resources. Mining methods include data stream adaptations for clustering methods, which group unlabeled data based on similar attributes; classification, which labels new data after training on existing labelled data; frequency counting to identify the most prevalent patterns; and time series analysis (Gaber, 2005).

Mining data streams using time series is particularly intuitive since time series and data streams possess significant theoretical similarities, according to Read (2020). After all, time series are simply a length of time-indexed data, whereas a data stream is a continuous data flow with members sequentially arriving individually or in groups; each tuple has several attributes that may or may not be time ordered. Therefore, data streams are a special instance of time series, and that tried-and-true statistical and dynamic systems time series methods can be applied. This means that ARIMA, partial differential equations, hidden Markov models, and recurrent neural networks can be successfully applied. However, stream data often exhibits time dependence, violating assumptions of statistical learning theory that guarantees solutions for the supervised machine learning methods of classification and regression. The relevance of classification methods even more questionable because there are very few data streaming scenarios in which data is labelled. Because of the continuous nature of most data stream features, regression can be successfully and readily applied (Read, 2020).

Geisler (2013) believed that DBMS designers must account for a variety of challenges specific to their application regarding data flow, distribution, and quality. Data quality is multifaceted: the percentage of empty tuples, volume of data used to calculate aggregations, age of the tuple, accuracy of mining algorithms, consistency of stream members, and the confidence that the data member values are correct should be considered (Geisler, 2013). Krempl (2014) identified the development of fully automatic preprocessing methods that optimize performance and evolve over time as another critical and difficult process, given that no one-size-fits-all preprocessing algorithms exist. Skewed distributions can also pose challenges: if there are binary features in which one outcome is far more probable than the other, it is difficult to track changes in the distribution over time. Finally, the continual drift of features over time means that convergence of machine learning algorithms might be guaranteed at one point in time, but not guaranteed later; it must be validated regularly, thus necessitating more resource budgeting.

Krempl (2014) also recognized that there are already many mining systems in place that employ these architectures to tackle the issues associated with data streaming. MobiMine was the first widespread system: it was used to analyzing stock market data. The user's mobile device would interact with a server to perform the necessary calculations. Subsequently, Karagupta and others developed Vehicle Data Stream Mining System to extract patterns from data streams generated by vehicles and analyze driver behavior with clustering in real time using hardware onboard the vehicle. Tanner and his team created the Environment for On-Board Processing to mine data streams generated by sensors onboard spacecraft, transmitting particularly interesting insights across the limited bandwidth back to earth (Krempl, 2014).

Although there has been significant advancement in the burgeoning field of data streams in recent years, Gaber (2005) still saw many opportunities for future research. These include the

minimization of sensor energy consumption, how to best preserve memory while maintaining performance, trend monitoring, avoiding overfitting, and effectively displaying stream mining results on mobile devices (Gaber, 2005). Krempl (2014) pointed to evaluation and accuracy tracking methods for data mining algorithms is another promising area for research; it is difficult to formulate heuristics for their development because data streams are so application specific and prone to drift. Moreover, developing processes to test learners is difficult; the traditional workflow of training, cross-validating, and testing is difficult to implement because streams are dynamic and time dependent (Krempl, 2014).

Preprocessing and Load Shedding

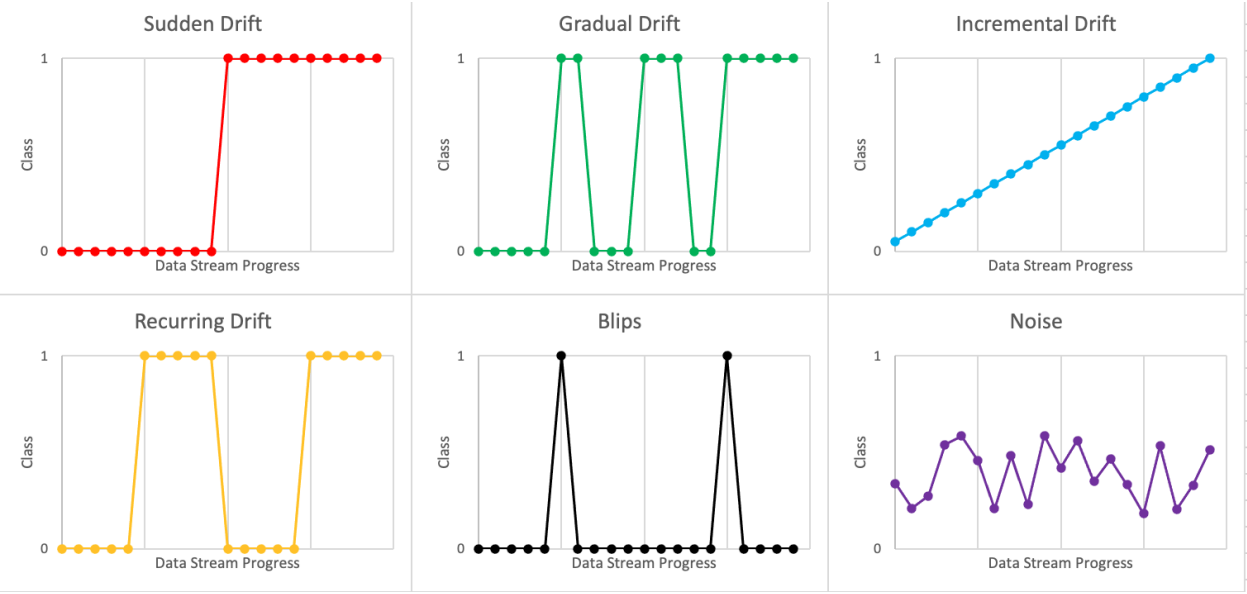
Data preprocessing is one of the more neglected aspects of data stream management; Ramírex-Gallego (2017) wrote that it requires over half of the total effort to implement and execute. Preprocessing aims to reduce the complexity of incoming information to allow for expedited, effective mining and general ease of data interpretation. This task entails removing noisy or superfluous features (akin to columns of data in a database) and instances (rows) from the sensor network. Depending on the application, preprocessing might also include the discretization, or bucketing, of continuous inputs. The process must account for rapid incremental arrival of data, unbounded volumes, limited time of access due to latency and memory requirements, limited or nonexistent access to labels, and potentially changing statistical distributions of features (Ramírex-Gallego, 2017).

Ramírex-Gallego (2017) labeled the presence of transient feature distributions as concept drift; preprocessing must identify drift involving substantive changes in the long-term central tendency and variance of the data and eliminate drift that does not significantly affect the long-term behavior of the feature. Drift is either local or global: local drift only affects a subset of the

feature space, whereas global drift impacts the entire set. Scope aside, drift can take one of six main forms. When subject to sudden drift, the behavior of a certain feature changes significantly and instantaneously. Gradual drift consists of oscillations between one tendency and another, with the old class becoming less prevalent and the new class becoming more prevalent until the new class entirely replaces the old. On the other hand, recurrent drift is the periodic, sudden, and sustained appearance of a new feature tendency; it is somewhat predictable, and the current tendency is still visible occasionally. Incremental drift is a continuous, steady shift from one behavior to another over time. Blips and noise are the final two types of drift: blips are dramatic, isolated outliers in standard behavior, whereas noise is random variation in the data that is not correlated to time (Ramírex-Gallego, 2017). See Figure 1 for a graphical representation of the various types of drift.

Figure 1

Types of Drift



Note: Graphical representation of the types of drift. This recreated figure is reprinted with the original publisher's permission (Ramírex-Gallego, 2017)

Ramírex-Gallego (2017) listed several ways to address the adverse impact of concept drift on learning models. For one, concept drift handlers measure feature behaviors such as standard deviation, stability, distribution, and learner accuracy. The handlers will alert the system when these metrics evidence a shift, at which point new machine learning models will be trained up on the data. As an alternative, systems can employ online learners that incorporate on each incoming instance. Ensemble learners are another option: this methodology involves training up a learner on the latest data, and, after a set time, comparing its performance against each member of a set of older learners. Learners in the committee are replaced to by new contenders if the new contestants' performance improves the quality of the ensemble; the best algorithm, or a combination of the best, are used as the final ensemble output (Ramírex-Gallego, 2017).

Sliding windows are also viable, according to Ramírex-Gallego (2017). They simply store a buffer of the most recent data for use in learning, forgetting older instances (Ramírex-Gallego, 2017). Galan (2005) adds that window size is bounded by memory constraints but should still be carefully considered based on the needs of the application and the behavior of the features: the larger the window, the smoother the learner's predictions will be. As the window size increases, predictions will become less vulnerable to noise, but less responsive to nonrandom changes. Splitting windows into smaller sub windows can decrease latency. The implementation details for this approach depend greatly on the application, and which statistics are most desirable to calculate quickly. For example, some approaches are ideal for returning counts and quantiles, whereas others are better at outputting best-fit slopes for learners (Galan, 2005).

Feature selection is one major subset of data preprocessing: Ramírex-Gallego (2017) defined its objective as removing unhelpful or unnecessary features, minimizing the size of the feature space while maintaining sufficient predictive accuracy in learners. The implementations of feature selection are conceptually like offline methods used for more static datasets, except they apply cumulative functions on statistics or other information to account for the continuous flow of instances. Features are either selected before delivery to the learner, or the functionality is integrated into the learner. One noteworthy consideration in the development of feature selection algorithms for streaming scenarios is the problem of a changing feature space: new features may be added, or existing ones may be removed. There are three primary approaches to this challenge. The Lossy Fixed method only considers the features in the first training batch, ignoring new features that become available as it synthesizes the learning model. Contrariwise, Lossy Local methods group the stream into training batches with corresponding test instances, generating a feature set for each training batch and its corresponding test instances. Lossless homogenizing, a technique that can work in conjunction with those mentioned above, creates a superset of all features that appear in the training and test sets for each batch and uses null values to fill space where no data existed before where necessary (Ramírex-Gallego, 2017).

Tatbul (2003) cited the optimization process of instance selection, or load shedding, as an integral part of an effective data stream management system. Quality of service must be maximized subject to the constraint of one or more bottleneck resources such as computational power, sensor bandwidth, sensor battery life, or database memory. Meeting these constraints might necessitate load shedding depending on the application. Instance selection is implemented using a drop operator, either random or semantic, that purges lines of data until a provided selectivity–a desired percent reduction in volume–is reached. Random drops accomplish the task

by simply eliminating instances at random until the selectivity criterion is met, whereas semantic drops eliminate the data with the lowest utility–utility calculations depend on the application. Either drop scheme can typically be wrapped in a data structure; operators can then invoke a predetermined, parametrized load shedding strategy on the database (Tatbul, 2003).

Galan (2005) explained that transforms achieve the results of instance selection using a different approach. Typically applied to time series, some transforms can be used to generate coefficients that represent a stream; the coefficients can serve as inputs to regression algorithms or assist in comparing streams. Fourier transforms and the creation of Haar wavelets–these have computational complexities of $O(n \log n)$ and $O(n)$ respectively–are candidates for streaming applications. However, they must operate on numerous points to become efficient. Moreover, piecewise aggregation approximation, which periodically saves the mean or median of a specific number of points and then discards the points themselves, is another viable option (Galan, 2005).

Preprocessing, Ramírex-Gallego (2017) realized, may also entail the discretization of continuous stream data. Equal-frequency discretization is the simplest incarnation: it continuously tracks the quantiles of the stream, using them to bucket the values. The Incremental Discretization Algorithm (IDA) implements this protocol by keeping a sample of the stream, using heaps to track the quantiles–heaps allow for the insertion or deletion of new elements in $O(n \log n)$ time. Other approaches can dynamically change the number of intervals: for example, the Partition Incremental Discretization algorithm will create an initial set of categories for the data, and then split single buckets or merge adjacent buckets when the number of elements in a category exceeds a splitting threshold or falls below a merging threshold (Ramírex-Gallego, 2017).

Aggregation

Zhang (2005) described aggregation as a critical aspect of data processing: summarization is an essential first step to glean insights from data. However, the traditional approach of writing every data point to a database and querying to produce statistics is infeasible, as the computation time of aggregation queries might not keep up with new arrivals. To counteract this, developers can use sliding windows, or cache data in structures such as heaps to calculate certain metrics. While the online production of aggregates is typically conceptually straightforward and intuitive, optimizing the querying infrastructure is a dynamic, continual, and challenging process. In many applications, analysts must perform numerous similar but slightly varied calculations to efficiently aggregate across complex, dynamic networks of sensors (Zhang, 2005).

Because resources are frequently limited, data volume and velocity can be immense, and the amount of computation required to digest the data is often formidable, Zhang (2005) understood streamlining the aggregation process to be critical. The Gigascope architecture is an example of one approach to this task. Gigascope operates by delegating the calculation of aggregates to two engines, a low-level processor to reduce volume and a high-level unit to create outputs. These modules are called the LFTA and HFTA respectively. The LFTA maintains hash tables of data fields. Fields reside in a hash table, with each value accompanied by a count of the number of times it has occurred. If the system attempts to insert a repeated value into the hash table, the count is incremented, but if a collision will occur, the old value in the hash table is evicted and moved to hash tables in the HFTA for querying. This phase of the process is the most expensive; Gigascope's main bottlenecks are probing the LFTA hash tables when new fields are inserted and evicting instances from the LFTA table to the HFTA tables (Zhang, 2005).

Once the data is moved over, Zhang (2005) continued that Gigascope maintains a hash table for each feature in the HFTA. Aggregation queries probe these tables to create outputs. To optimize this process, Gigascope also uses phantoms, which allow for shared computation of similar aggregations where the same metric must be calculated for different columns that are frequently accessed together. Phantoms accomplish this by merging the hash tables that represent the features of interest to allow for simultaneous computation of the multiple aggregates. To support these processes, Gigascope will run two optimizations. Firstly, it efficiently allocates its memory allowance: Gigascope utilizes all the memory it is given to enlarge the hash tables for the purpose of reducing the number of collisions, which are computationally expensive. Second, it evaluates which phantoms are beneficial by comparing overall hash table maintenance costs with and without a variety of different phantoms (Zhang, 2005).

There are several requirements associated with summarizing data across sensors. Henning (2020) emphasized that sensor networks are often multi-layered: depending on the application, statistics from individual sensors may need to be aggregated into groups, which the system may subsequently combine with data from other groups. After all, sensor data might be merged or linked in multiple parallel hierarchies because it may be collected and aggregated at multiple levels at numerous production facilities within a company. The grouping systems must run constantly, handling sensor addition or removal from the network and group structure reformatting (Henning, 2020).

A dual streaming system can augment aggregation across sensors, Henning (2020) asserted. The process considers two sources of data: the input stream, and a table that maps individual sensors to the groups they belong in. As new instance batches arrive, the system merges them with historical aggregate data, and then joins the resulting stream with the sensor

group table to propagate the impact of the new data through the hierarchy the sensor is associated with. After the merging and integration, the program then duplicates the newest measurement once for each ancestor group. Each copy turns into a key value pair where the key is a unique combination of the sensor ID and a group ID. If a sensor is removed from a group between iterations, a tombstone value replaces the key value pair. The system then pipes the copies into a last value table, which contains the last recorded value for each sensor and group. From there, the system groups the last value table by the group identifier portion of the entries' keys and performs aggregations on the groups. It then publishes the aggregation results in a stream to be merged with new data arrivals in the next iteration of the process (Henning, 2020).

Although powerful, Henning (2020) qualified that this basic dual streaming architecture is not robust to records that arrive out of order. Tumbling or hopping windows can resolve this issue by splitting the timeline up into windows and storing additional information in the last value table. A tumbling window splits the timeline up into sequential windows of fixed, constant size and stores the last recorded value for each time window within each sensor group in the last value table. Old windows are discarded after a certain application specific amount of time is reached. Instead of publishing aggregation results after each instance arrival, a tumbling window system submits results at the close of each window. The window size must be determined in advance and should equal the maximum expected time between measurements. In this implementation, the emit rate–the speed at which aggregates are submitted–is static, predetermined, and based on the frequency of sensor measurements. A hopping window implementation, on the other hand, overcomes some of these difficulties by allowing analysts to select the rate at which aggregation results are published independently of the sensor measurement frequency. To support this, the system uses a tumbling window architecture with

one important exception: instead of being sequential, the windows are allowed to overlap when the desired emit rate is not an integer multiple of the tumbling window size. The system still publishes results at the end of each window, but measurements can belong to more than one window (Henning, 2020).

Storage and Querying

Golab (2014) wrote that the demands of streaming scenarios necessitate specific infrastructure to ensure data quality and the timeliness of responses in the storage and querying systems. Firstly, the system should store application specific dimensional data–information about the entity properties–both past and present (Golab, 2014). To guarantee that the system can execute queries in well-defined time intervals, Diallo (2012) corroborates that it will maintain certain supporting metrics and metadata for each instance, transaction, and sensor. Each measurement must have a timestamp, as well as an absolute validity interval denoting the period over which the values will be considered relevant. In addition, there must be infrastructure to determine and track certain information about each query on the data, including the liberation time at which all requisite resources are free to perform the calculation, the computation time necessary for the operation, and the maximum time allowable for the process. Spatial data about the current and historical locations of the sensors is also of interest (Daillo, 2012).

Diallo (2012) exposited several existing methods that can maintain this overhead, including PoTree, PasTree, and StH. All three utilize tree-based structures; the PoTree contains two subtrees, one for spatial data and one for temporal information. PasTree refines the PoTree architecture, whereas StH builds on both by maintaining basic data about the sensors, such as ID and historical positions. StH also provides capabilities to determine when to move stored data

form sensor networks to more permanent warehouses, using a heat function to track instance freshness (Daillo, 2012).

Golab (2014) explained that the need for timeliness and quality informs other requirements and methods. Quality monitoring often employs integrity constraints on incoming and stored instances; if the constraints are violated, the system will take necessary action, or inform its owner of an issue. To optimize analytic performance, data streaming applications often employ fast ETL (extract, transform, and load). Moreover, many architectures partition the data into sections by timestamp to allow for easier reading and writing. Many systems will merge incoming out of order tuples into existing partitions, maintain variable partition sizes so that older instances reside in larger buckets, and transform section structure from write optimized to read optimized as aging occurs. Every effective data stream management system must dynamically maintain materialized views–saved, commonly used queries–rather than recalculating them all at once. After all, the views must be constantly accessible, and the data volume and velocity often prohibit all-at-once recalculation. Centralized storage and processing servers must receive new instances continuously rather than at downtime, schedule updates to ensure freshness and combat overloading, and can delegate responsibilities across machines to improve processing (Golab, 2014).

Daillo (2012) stated that there are two primary architectures for storing and querying: distributed and warehousing. A distributed approach attempts to limit the amount of data transfer by taking advantage of the storage capacities and computational power of individual sensors. Rather than sending all incident stream data to a central location, sensors in a distributed stream management architecture perform use their inbuilt processors to perform some aggregation and load shedding. A distributed architecture stores data both on the individual sensors and on a

central server, allowing one to query either or both. Sensors in a distributed network process both short-term queries run against them that can return results quickly, and long-term queries that operate on incident tuples. Only queries on older historical data are delegated to the server. The server receives processed data and older instances for more permanent residence (Daillo, 2012).

Golab (2014) explained warehousing as an alternative conceptually like a typical database management system in many ways, or a typical data stream management system with robust historical records. One of the primary dissimilarities between a data stream warehouse and a database management system is that the former must support continuous updates with no downtime, whereas many typical databases are shut down for updates overnight. Moreover, the data warehouse should continually provide insights and recommendations regarding the workflow it augments. The system must also adequately handle late, out of order, missing, duplicate, and incorrect instances. Warehouses are typically constructed in one of three ways: adding real time loading and querying functionalities to a traditional database management system, supplementing a data stream management system with persistent storage, or modifying an analytics system to allow for real time processing. To implement queries and other desired capabilities, warehouses use a variety of languages such as sliding window based streaming SQL, event-oriented scripting languages like DejaVu, AQuery for sequencing, and Nova for workflow control (Golab, 2014).

Although researchers have made significant improvements and advances in developing data stream management architectures, Golab (2014) observed many open opportunities to better leverage modern technology for data stream management systems remain. For example, although they utilize the space and processing of sensors and servers, streaming systems typically fail to capitalize on the main memory and secondary storage available on client computers.

Furthermore, most warehousing takes place on servers, when it could be conducted more effectively on the cloud. Warehousing systems are also awaiting a more suitable integration of stream mining and machine learning into their set of capabilities. Another promising frontier for further work lies in the exploration of novel, hybridized streaming architectures that combine the strengths of numerous resources and methods to produce a more optimal system (Golab, 2014).

Data Mining and VFML

Zenisek (2022) argued that applying machine learning to data streams can add value to organizations in numerous ways. For example, it can assist in scheduling preventative maintenance and estimating the remaining useful life of the equipment and processes that sensors monitor. Moreover, it is exceedingly useful in merging data streams with no clear key, such as a timestamp, to join on. Without the application of machine learning for stream merging, subject matter experts must construct time windows for combining the streams. However, classification algorithms can train on expert decisions to automate the process. This has significant repercussions, allowing the features of multiple sensor streams to be mathematically combined to create virtual sensors which can approximate the readings of sensors that are not feasible to physically implement. The dynamic merging can also assist with process control, synthesizing industrial equipment data with production line sensor data to allow the equipment to self-correct without downtime (Zenisek, 2022).

Santini (2006) wrote that machine learning can also reduce network bandwidth consumption by decreasing the frequency of communication between sensor sources, intermediate devices, and the server sink. The goal of this approach is to minimize transmissions between the sensor and server while ensuring that the stream complies with a user-specified accuracy value. Researchers have presented numerous solutions for this problem; one of the

earlier suggestions entailed representing sensor readings as an optical image, performing video compression on the sensor, and transmitting the result to the sink for decoding. Analysts have subsequently proposed approaches optimized for various performance criteria such as convergence rate, general robustness, computational load, and more. For example, the kernel linear regression and dual Kalman filter models both maintain predictive models of the data stream at the source and sink (the former uses kernel linear regression whereas the latter runs Kalman filters at the sensors), only sending updates from the source to the sink when the deviation of the stream from the prediction exceeds a maximum acceptable user error. Although the dual Kalman filter architecture excels in noisy environments, both architectures require a priori knowledge about the statistical distribution governing the stream to initialize the predictive models (Santini, 2006).

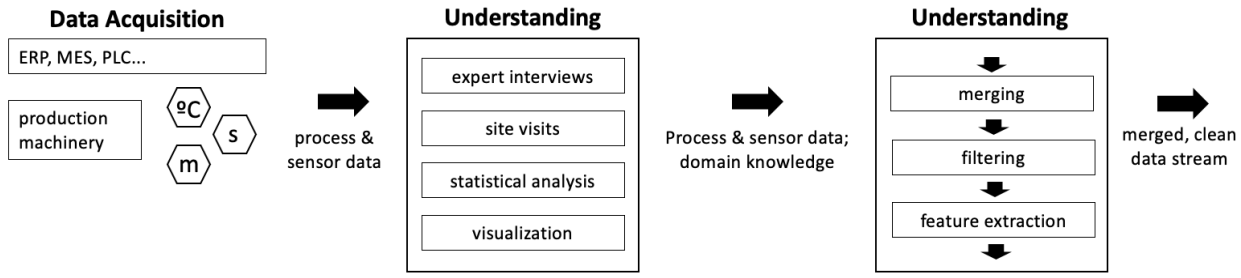
Santini (2006) also discussed more dynamic solutions such as the Least Mean Squares (LMS) algorithm, which perform the same task without requiring understanding of the underlying statistical distribution. The LMS algorithm operates in three modes. The first is initialization, in which the source and sink nodes are in constant communication as they build up the predictive models. Once these models are built, the system either operates in normal mode, in which the source gathers a given number of readings and then sends updates to the model coefficients based on the prediction error, or standalone mode, where the source discards readings that lie within acceptable prediction error. The model switches from standalone mode to normal mode if the prediction accuracy becomes unacceptable, and back to standalone mode if the accuracy sufficiently improves (Santini, 2006).

Deriving insights, implementing controls, and conserving resources using machine learning is clearly beneficial, Zenisek (2022) posited; the methods and techniques presented in

prior sections lay the groundwork and provide the prerequisite tools for data mining. In preparation for mining, analysts must first understand the data with expert interventions, summary aggregations, and visual tools. This readies the stream for final preparations, including merging, load shedding, and feature selection. Once the instances have been treated in this manner, mining can commence: the data stream management system can apply regression and classification models, test them to measure their accuracy, apply the insights gained from the algorithms to manufacturing processes, and deploy the system to company cloud or local infrastructure (Zenisek, 2022). See Figure 2 for an illustration of the process of data mining preparation, and Figure 3 for a depiction of the data mining process.

Figure 2

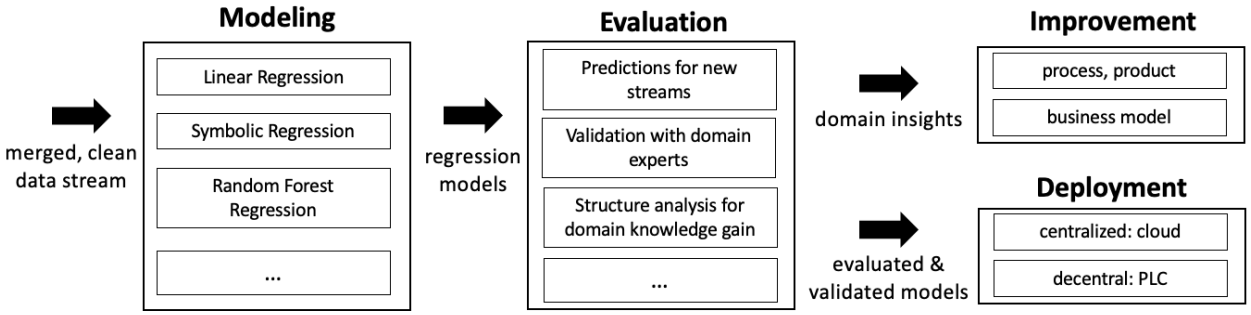
Data Mining Preparation



Note: The process of readying data for mining. This recreated figure is reprinted with the original publisher's permission (Zenisek, 2022)

Figure 3

The Data Mining Process



Note: A summary of the mining process. This recreated figure is reprinted with the original publisher's permission (Zenisek, 2022)

Gaber (2012) expounded several categories of general techniques for implementing data stream mining. The first type are two phase methods, which start off by running microclusters. These microclusters run in real time on incoming instances, generating summary of the stream. In the second phase, an offline module will periodically run machine learning algorithms on the microclusters' summaries. The details of this operation depend largely on whether the end goal is to apply a supervised or unsupervised learner (Gaber, 2012).

Gaber (2012) also described Hoeffding bound methodologies, which are built the calculated upper bound of a learner's accuracy loss as a function of the number of data records at each algorithmic iteration. This information can be used to extend clustering learners and decision trees to data streams, addressing concerns regarding ties of splitting attributes for trees, data velocities, limited memory, and output accuracy. Hoeffding bounds can resolve ties in the splitting criteria by using a user inputted acceptable error threshold to enforce a limit on the number of instances the system can view for the step. Once the system analyses the maximum number of allowed records to resolve the tie between splitting attributes, it must choose the one

with the slightly more favorable splitting criterion value or decide arbitrarily. Likewise, Hoeffding bound based decision trees address data velocity by processing instances in groups, instead of as individuals. They conserve space in the system by periodically checking the tree to delete leaves and splits that no longer provide sufficient value. Finally, Hoeffding bound decision trees ensure accurate results by initializing the tree intelligently, and by passing over new data multiple times if the arrival rates allow (Gaber, 2012).

According to Gaber (2012), Symbolic approximation (SAX) represents a data stream as a time series and is especially useful for identifying the most frequent and most different subsequence–called the motif and discord respectively–in the set. SAX accomplishes this task using a trifold plan. In the first step, it resizes the time series based on an equation output. Then it discretizes the time series into chunks indexed by a character. Finally, it applies a function to find the distance between each bucket (Gaber, 2012).

In addition to these three implementation strategies for various data mining algorithms, Gaber (2012) showed how algorithm granularity can be used to support processing. Granularity is the practice of dynamically varying an algorithm's inputs, outputs, and processing based on resource availability and data rates. A system can enforce input granularity by varying the selectivity of load shedding and sampling, along with aggregation policies. These factors are based on application specific bottlenecks such as system memory, data velocity, network bandwidth, sensor battery life, and arrival volumes. Likewise, it achieves output granularity by dynamically expanding or reducing the size of the algorithm output based on available. Finally, an architecture can employ processing granularity by modifying algorithm subroutines based on limited processing power, using randomization and approximation if the CPU load does not allow for full precision (Gaber, 2012).

Researchers have employed these concepts and mechanics, along with others, to assist in synthesizing regression methods fit for data streaming applications. Fast and Incremental Trees (FIMT-DD), Gomes (2018) wrote, are one of the more popular approaches and are similar in many ways to Hoeffding trees. Gathering stream statistics for a given time interval, FIMT-DD then ranks features by variance. Comparing the two highest ranked, the tree will split off two new branches if the two features differ by more than the Hoeffding bound. ORTO works comparably, although it provides option nodes that allow instances to run through all branches of a tree node. Moreover, regression rule-based techniques are useful in streaming. For example, the Adaptive Model Rules (AMRules) algorithm builds an ordered and unordered rule set; each rule is supervised by a drift detector that monitor the rules using feedback from the stream (Gomes, 2018).

Other methodologies use ensembles of learners for prediction to help proactively combat and reactively recover from concept drift (Gomes, 2019). Gomes (2018) elaborated in another work that Scale-Free Network Regression prepares several learners in a probabilistic scale-free grouping where more accurate learners have more impact in the prediction step. The Adaptive Random Forest (ARF-Reg) algorithm combines the power of ensemble learning with the strengths of the FIMT-DD in a feature rich platform. ARF-Reg leverages voting by aggregating individual learner outputs into a final prediction and incorporates diversity by providing each model with unique training and test data. Leveraging the efficient splitting and feature selection of the FIMT-DD program by using it as the base learner, ARF-Reg combats drift by using drift detectors to throw warnings. When the detectors signal warnings, the system handles by training up new learners in the background to replace ones that are becoming invalid (Gomes, 2018).

Statistical Process Monitoring and Data Streams

Qiu (2017) thought that Statistical Process Control (SPC), along with the control charts it provides, are promising data stream monitoring strategies. At present, it is largely used for production lines in both phase 1 and phase 2 applications. In phase 1 applications, a facility team is attempting to establish a line, so they produce test batches of material and monitor certain critical to quality variables on control charts. If the values of the quality variable lie within the control limits of the chart over a sufficiently large sample size, the process is in control and phase 2 monitoring can commence. Phase 2 SPC consists of online process monitoring, and its objective is to signal that a line is out of control as quickly as possible (Qiu, 2017).

SPC tools include the X-bar chart, which flags a process as out of control if a quality variable differs from the mean by an amount that lies within the tail of a normal distribution with a specific area, Qiu (2017) told readers. This area corresponds to half of an application specific statistical parameter known as the significance level. The significance level is typically chosen based on average run length, the amount of time between a mean shift and an appropriate in control or out of control signal. Cumulative sum charts are also valuable: these compare the running sum of all the data points within a given window to an upper and lower control limit chosen based on the desired average run length. Additionally, exponential weighted moving averages are designed to track the central tendency of a process (Qiu, 2017).

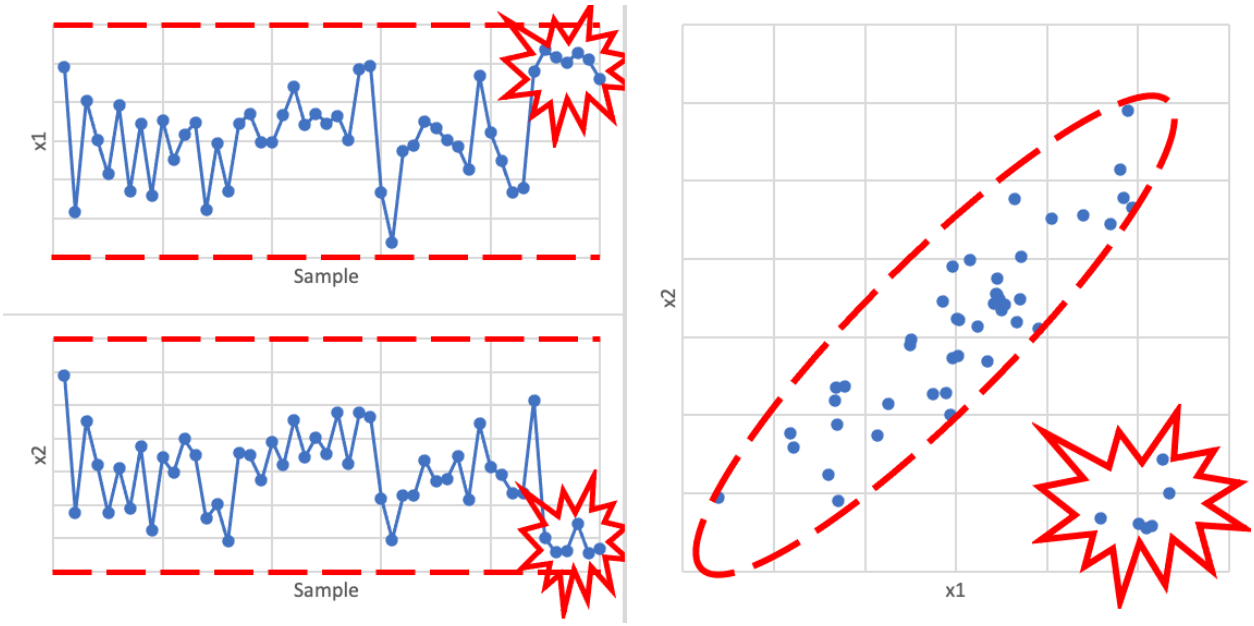
Basic SPC charts assume a constant process mean and variance, which Qiu (2017) recognizes is not the case in many streaming scenarios. The dynamic screening system (DySS) method addresses this issue in three steps, allowing for the extension of SPC charts for monitoring processes with time-varying statistical distributions. First, it estimates the change in central tendency and variance of the quality variable with time using a suitable in control dataset.

Secondly, the algorithm standardizes incoming observations based on this pattern. Finally, the program uses traditional SPC to monitor the standardized data, indicating significant shifts of the incident data's mean and variance from the recorded pattern (Qiu, 2017).

Although SPC is a powerful tool for tracking single quality variables, He (2018) denounced the ability of traditional methods to trace the correlations between quality variables and signaling substantive changes in the relationship. Multivariate Statistical Process Monitoring (MSPM) excels in this regard, using techniques like principle component analysis, partial least squares, and multivariate graphical analysis to identify lurking variables. However, it is hampered by assumptions of normality, latency free data transfer, time-independence of distributions, and linear relationships between variables; it is not robust to outliers, errors, or failed sensors. Many algorithms have been proposed to prevent MSPM's reliance on these assumptions, and data preprocessing can protect MSPM against outliers and noise (He, 2018). See Figure 4 below for a set of graphs that show he need for MSPM.

Figure 4

The MSPM Value Proposition



Note: An example of the MSPM value proposition: a pair of X bar charts (left) fail to identify a deviation from a correlation which is clearly detected by a MSPM tool (right). This recreated figure is reprinted with the original publisher's permission (He, 2018)

Correlation of multiple data stream variables–e.g., input and output features–across time is a factor Sayal (2004) found important. Correlation rules describe four aspects of the relationship between variables: direction, sensitivity, delay, and confidence. Direction indicates whether the output increases or decreases as the input increases. Sensitivity specifies how much impact a given change in an input has on an output. Delay, on the other hand, is the time interval between a change in an input feature and the corresponding shift in the output. Finally, confidence reports the degree to which analysts are certain a correlation is indeed present (Sayal, 2004).

Sayal (2004) proposed that data stream correlation analysis consists of first aggregating the instances from the various streams at several granularities (seconds, minutes, hours, etc.), and then detecting change points on the graph of the cumulative summary of the various streams. This graph documents the cumulative sum of the stream on the y-axis, where the value of the mean or median is subtracted from each point; the values where the resulting graph crosses the x-axis are important. Applying the cumulative sum discretizes continuous data for ease of use and allows analysts to focus on timestamps where significant changes occur. Once the cumulative sums are observed, analysts can merge the data streams into a single time series by convolution using a mathematical function, or by a simple arithmetic sum of the features (Sayal, 2004).

With the data in a single time series, Sayal (2004) said that analysts can compare features to construct correlation rules. Time correlation is assessed by finding the maximum covariance of the two features of interest, divided by the product of the individual standard deviations of the features across all possible time intervals. Delay corresponds to the time difference that yields the maximum correlation. Using this, analysts can determine the sensitivity and direction of the correlation by comparing the change in the input and output features across a time interval with a duration equal to the correlation delay. Finally, they can assess confidence by calculating the percentage of times that the statistical correlation–covariance over product of standard deviations–across the delay is above a user defined threshold value (Sayal, 2004).

Managing Sensor Data Streams in a Quality Control Department

Although there is a significant body of research on data streams and applications thereof, there is little work on making the most of sensor data in industrial quality control settings. This gap in the literature shall be addressed with a novel, concrete methodology that will derive necessary insights from stream data without excessive memory or computation needs.

Aggregating sensor data, along with potential process input variables, at various granularities is the first step in deriving meaningful insights. The aggregation intervals will depend on the velocity of sensor data: if a sensor of interest takes a measurement several times per second, data could be aggregated by second, minute, and hour. Depending on the counts for each aggregate, various statistics measuring variance and central tendency of important variables could be stored, such as standard deviation, mean, and range. The individual measurements are not of particular interest; in a manufacturing setting, consecutive sensor readings taken at high frequencies should not be significantly different. Even if they are, large changes will still be identified as soon as the next aggregate is calculated. As the aggregates at various levels are calculated, their measures of central tendency can be plotted on relevant control charts such as x bar and s bar. As time goes by, the system should implement a policy to delete low level aggregates based on the passage of time and the severity of memory constraints. For example, aggregates by second might be dropped a day after collection, and the data can be described by minute aggregates; a month after the data are collected, the minute aggregates might be dropped, so the data are described by hourly aggregates, and so on.

These aggregates can provide meaningful insights with low resource overhead, as old data is summarized with decreasing granularity. Firstly, the system can flag and retain aggregate timestamps corresponding to defective or nonconforming process outputs, notifying relevant quality assurance and control personnel for swift corrective action. Secondly, as the sensors read in data and the system aggregates it, the system can also utilize aggregated values from the data streams of potential process inputs over the same time intervals, or the static values of possible inputs that are more stable. The system can analyze these input variables in conjunction with the sensor data streams by continuously performing and updating regression models and assessing

time correlations to determine the latency of the regression relationship. For example, when a process input variable changes, perhaps the corresponding fluctuation in the sensor data comes several minutes later.

Depending on the application scenario, the data velocity, the computational and memory resources that are available, and the desired level of specificity and granularity, it may be feasible and advisable to monitor individual sensor measurements rather than frequent aggregates. In this approach, observations could be plotted on X-bar control charts for individual observations, MR control charts, or CUSUM control charts. Nonconforming or defective observations can still be timestamped and flagged, and the system can still notify quality control personnel. Moreover, the system can still implement memory conservation protocols by aggregating the individual observations after a certain period has passed, storing the aggregates, and dropping the individual observations. As time passes, the system can decrease the granularity of old aggregates to conserve space.

Both approaches can, and should, be supplemented by additional methods discussed in this work. Incident streams should be preprocessed to ensure the data quality of inputs; if a sensor transmitting data is not functioning properly, its measurements should be ignored. Moreover, if sensor bandwidth is a process bottleneck or a significant cost, then analysts should consider leveraging the processing and storage capabilities of the sensors by utilizing distributed queries. To reduce transmissions further, users can configure regression models on both the centralized server and the individual sensors; instead of transmitting periodic aggregates or continuous measurements from the sensor to the server, the sensor can simply contact the server when its predictive model for the data stream reaches an application specific error threshold. At

this point, the sensor can provide an updated model, or begin sending the server individual measurements so the server can adjust the regression weights and features itself.

Conclusion

In summary, quality control in modern manufacturing necessitates the use of sensors, which lead to the accumulation of massive amounts of information in company database infrastructure. Orthodox database management systems and strategies often cannot cope with the high volume and velocity of these data streams. Responding to this important and overlooked issue, this work reviews techniques for handling data streams in a manufacturing quality control setting. Firstly, it discusses existing DSMSs, focusing on their basic common architectural components, and their general approaches to dealing with data stream volume and velocities. Citing historical and modern data stream management systems, the work introduces preprocessing and aggregation. Preprocessing addresses glaring data quality issues including noise, outliers, and missing tuples, whereas aggregation creates statistical summaries of the data to relax system memory requirements and make important trends and changes easier to identify. Next, the work exposits the tree-based architectures used for storing data tuples, and the distributed and centralized heuristics for storing and querying the information. Then it turns to VFML techniques that are useful for merging streams without clear key attributes to join on, and for classification and regression tasks operating on streams. Subsequently, it introduces statistical process monitoring in and some tools to adapt control charts to streaming scenarios. Finally, a novel, adaptable, high-level strategy for utilizing data streams for quality control in manufacturing without excessive memory burdens is provided. This approach is applicable in a wide variety of industries and use cases, and relies on the DSMSs, preprocessing techniques, machine learning algorithms, and statistical process monitoring tools presented throughout. Case

studies utilizing this approach, along with tested data stream management practices, in quality control departments, are a promising area for future scholarship.

References

Bifet, A., Zhang, J., Fan, W., He, C., Zhang, J., Qian, J., Pfahringer, B, et. al. (2017, August). Extremely fast decision tree mining for evolving data streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1733-1742).

Diallo, O. R. (2012). Real-time data management on wireless sensor networks: A survey. *Journal of Network and Computer Applications, 35*(3), 1013-1021. https://doi.org/10.1016/j.jnca.2011.12.006.

Gaber, M. M. (2005, June). Mining data streams: A review. *SIGMOD Rec., 34*(2), 18-26. https://doi.org/10.1145/1083784.1083789

Gaber, M. M. (2012, January). Advances in data stream mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2* (1), 79-85. https://doi.org/10.1002/widm.52

Galan, M., Liu, H., & Torkkola, K. (2005). Intelligent instance selection of data streams for smart sensor applications. *Procedings of SPIE - The International Society for Optical Engineering, 5803.* https://doi.org/10.1117/12.605855

Geisler, S. (2013). Data stream management systems. In M. L. Kolaitis, *Data Exchange, Integration, and Streams* (Vol. 5). Dagstuhl Follow-Ups. ISBN: 978-3-939897-61-3.

Golab, L. & Johnson, T. (2014). Data stream warehousing. *International Conference on Data Engineering.* Chicago, IL: IEEE. https://doi.org/10.1117/12.60585510.1109/ICDE.2014.6816763

Gomes, H. M., Barddal, J. P., Ferreira, L. E. B., & Bifet, A (2018). Adaptive random forests for data stream regression. *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning.* Belgium.

Gomes, H. M., Read, J., Bifet, A., Barddal, J. P., & Gama, J. (2019). Machine learning for streaming data: State of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter*, *21*(2), 6-22.

He, P. & Wang, J. (2018, July). Statistical process monitoring as a big data analytics tool for smart manufacturing. *Journal of Process Control, 67*, 35-43. https://doi.org/10.1016/j.jprocont.2017.06.012

Henning, S., Hasselbring, W. (2020). Scalable and reliable multi-dimensional sensor data aggregation in data streaming architectures. *Data-Enabled Discov. Appl.* 4, 5. https://doi.org/10.1007/s41688-020-00041-3.

Krempl, G. et al. (2014, June). Open challenges for data stream mining research. *ACM SIGKDD Explorations Newsletter, 16* (1), 1-10. https://doi.org/10.1145/2674026.2674028

Olken, F., & Gruenwald, L. (2008). Data stream management: Aggregation, classification, modeling, and operator placement. *IEEE Internet Computing, 12*(6), 9-12. https://dx.doi.org/10.1109/MIC.2008.121

Qiu, P. (2017). Statistical process control charts as a tool for analyzing big data. In S. E. Ahmed, *Big and Complex Data Analysis: Methodologies and Applications* (pp. 123-138). https://doi.org/10.1007/978-3-319-41573-4_7

Ramírez-Gallego, S., Krawczyk, B., Garcia, S., Wozniak, M., & Harrera, F. (2017, May 24). A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing, 239*, 39-57. https://doi.org/10.1016/j.neucom.2017.01.078

Read, J., Rios, R. A., Nogueira, T., & de Mello R. F. (2020). Data streams are time series: Challenging assumptions. *Brazillian Conference on Intelligent Systems*, (pp. 529-543). https://doi.org/10.1007/978-3-030-61380-8_36

Santini, S., & Romer, K. (2006, June). An adaptive strategy for quality-based data reduction in wireless sensor networks. In *Proceedings of the 3rd international conference on networked sensing systems (INSS 2006)* (pp. 29-36). Chicago, IL: TRF.

Sayal, M. (2004). Detecting time correlations in time-series data streams. HP Technical Reports. Retrieved February 28, 2023, from https://www.hpl.hp.com/techreports/2004/HPL-2004-103.html.

Tatbul, N., Cetintemel, U., Zdonik, S., Cherniack, M., & Stonbraker, M. (2003). Load shedding in a data stream manager. In *Proceedings 2003 VLDB Conference*, (pp. 309-320). https://doi.org/10.1016/B978-012722442-8/50035-5.

Zenisek, J. G. (2022). Machine learning based data stream merging in additive manufacturing. *Procedia Computer Science, 200*, 1422-1431. https://doi.org/10.1016/j.procs.2022.01.343

Zhang, R. K. (2005). Multiple aggregations over data streams. *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 299-310. https://doi.org/10.1145/1066157.1066192