# BIG-DATA STREAMS CLASSIFICATION THROUGH NEAREST NEIGHBOUR

Nagashri J A
M.Tech Student
Department of Studies in Computer Science & Engineering,
University BDT College Of Engineering
(A Constituent College OF VTU, Belagavi)
Davanagere, Karnataka.

B N Veerappa
Associate Professor & Chairman
Department of Studies in Computer Science & Engineering
University BDT College Of Engineering
(A Constituent College OF VTU, Belagavi)
Davanagere, Karnataka.

***Abstract :*** Massive and high-speed data streams among the main contemporary challenges in machine learning. This calls For methods displaying a high computational efficacy, with ability to continuously update their structure and handle ever-arriving big number of instances. In this paper, we present a new incremental and distributed classifier based on the popular nearest neighbor algorithm, adapted to such a demanding scenario. This method, includes a distribute diametric-space ordering to perform faster searches. Additionally, we propose an efficient incremental instance selection method for massive data streams that continuously update and remove outdated examples from the case-base. This alleviates the high computational requirements of the original classifier, thus making it suitable for the considered problem. Experimental study conducted on a set of real-life massive data streams proves the usefulness of the proposed solution and shows that we are able to provide the first efficient nearest neighbor solution for high-speed big and streaming data.

***IndexTerms*** **-Big data, data streams, distributed computing, instance reduction, machine learning, nearest neighbor.**

## I. INTRODUCTION

THE massive volume of information gathered by contemporary systems became omnipresent, as many research activities require collecting increasingly huge amounts of data. For instance, Large Hadron Collider experiments1 generates 30 petabytes of information per year. Potential applications for massive data analysis techniques could be found in each human activity domain. Enterprises would like to discover interesting client behavior characteristics, e.g., on the basis of sensor or Internet data. Works on personalized medical treatment for individual patients based on his/her clinical records, such as medical history, genomic, cellular, and environmental data may serve as another example. We are surrounded by enormous volumes of data arriving continuously from different sources. Therefore, one may say that we are living in the *big data era*. Big data is usually characterized by the so-called 5V's (volume, velocity, variety, veracity, and value), describing its massive volume, dynamic nature, diverse forms, different qualities, and usefulness for human beings.

In many cases we do not deal with static data collections, but rather with dynamic ones. They arrive in a form of continuous batches of data, known as data streams. In such scenarios, we need not only to manage the volume but also the velocity of data, thus constantly updating and adapting our learning. To add a further difficulty, many modern data sources generate their outputs with very short intervals, thus creating the issue of high-speed data streams. Massive data must be explored efficiently and converted into valuable knowledge which could be used by enterprises (among others) to build their competitive advantage.

However, there exist a considerable gap between contemporary processing and storage capacities, which demonstrates that our ability to capture and store data has far outpaced our ability to process and utilize it. Moore's law says that processing capacity double every 18 months, while disk storage capacity doubles every 9 months (storage law). This leads to creation of the so-called *data tombs*, i.e., volume of data which are stored but never analyzed. Therefore, we have to develop dedicated tools and techniques which are able to mine enormous volumes of incoming data, while additionally taking into consideration that each record may be analyzed only once to reduce the overall computing costs. Map Reduce was the first programming paradigm designed to deal with the phenomenon of big data. Recently, a new large-scale processing framework, is gaining importance in the big data domain due to its good performance in iterative and incremental procedures. Lazy learning (also called instance-based learning) is considered as one of the simplest, yet most effective schemes in supervised learning. Here, generalization is deferred until a query is made to the case-base.

As distance between every pair of cases must be computed (quadratic complexity), these methods tends to have much slower classification phase than their counterparts. Furthermore, lazy learners, as *k*-nearest neighbor (*k*-NN), tend to accumulate instances from data streams, thus leading to using data related to the outdated concepts may for the decision-making process. As of these reasons lazy learning has not been widely used in streaming environments in spite of its attractive properties. Data

reduction techniques may be applied to improve the performance of lazy learners. Concretely, instance selection techniques can be very effective as they reduce the total number of samples stored in the case-base and therefore simplify the underlying search space. Search speed may also be improved by introducing an implicit metric-space ordering in the case-base or through other techniques as locality-sensitive hashing.

In this paper, we propose an efficient nearest neighbor solution to classify high-speed and massive data streams. Our algorithm consists of a distributed case base and an instance selection method that enhances its performance and effectiveness. A distributed metric tree has been designed to organize the case-base and consequently to speed up the neighbor searches. This distributed tree consists of a top-tree (in the master node) that routes the searches in the first levels and several leaf nodes (in the slave's nodes) that solve the searches in next levels through a completely parallel scheme. Performance is further improved by a distributed edition-based instance selection method, which only inserts correct examples and removes the noisy ones. Up to the best of our knowledge, this is the first lazy learning solution in dealing with large-scale, high-speed, and streaming problems.

The main contributions of this paper are as follows.
1) Efficient and scalable incremental nearest neighbor classification scheme for massive and high-speed data streams.2) Smart partitioningof the incoming data streams. 3) Embedded instance selection method with quickly updated hybrid trees. 4) Comprehensive experimental evaluation of the proposed methods. Experimental results performed using several datasets and configurations show that our proposal outperforms the same model without edition in terms of accuracy. Our method also reduces the time spent in the prediction stage and the memory consumption.

## II. LITERATURE SURVEY

The term 'Big-Data' has spread rapidly in framework of Data Mining and Business Intelligence. This new scenario canbe defined by means of those problems that cannot be effectively or efficiently addressed using the standard computing resources that we are currently have. We must emphasize that Big Data does not just imply large volumes of data but also the necessity for scalability, i.e., ensure a response in an acceptable elapsed time. When the scalability term is considered, usually traditional parallel-type solutions are contemplated.

In many domains, data now arrives faster than we are able to mine it. To avoid wasting this data, we must switch from the traditional \one-shot" data mining approach to systems that are able to mine continuous, high-volume, open-ended data streams as they arrive. In this extended abstract we identify some desiderata for such systems, and outline our framework for realizing them. A key property of our approach is that it minimizes the time required to build a model on a stream, while guaranteeing (as long as the data is i.e.d) that the model learned is effectively indistinguishable from the one that would be obtained using infinite data. Using this framework, we have successfully adapted several learning algorithms to massive data streams, including decision tree induction, Bayesian network learning, k-means clustering, and the EM algorithm for mixtures of Gaussians. These algorithms are able to process on the order of billions of examples per day using o_-the-shelf hardware. Building on this, we are currently developing software primitives for scaling arbitrary learning algorithms to massive data streams with minimal effort.

In many domains, the massive data streams available today make it possible to build more intricate (and thus potentially more accurate) models than ever before, but this is precluded by the sheer computational cost of model-building; paradoxically, only the simplest models are mined from these streams, because only they can be mined fast enough. Alternatively, complex methods are applied to small subsets of the data. The result (we suspect) is often wasted data and outdated models. In this extended abstract we outlined some desiderata for data mining systems that able to \keep up" with these massive data streams, and some elements of our framework for achieving them.

Map Reduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. Our implementation of Map Reduce runs on a large cluster of commodity machines and is highly scalable: a typical Map Reduce computation processes many terabytes of data on thousands of machines. Programmers and the system easy to use: hundreds of Map Reduce programs have been implemented and upwards of one thousand Map Reduce jobs are executed on Google's clusters every day.

The Map Reduce programming model has been successfully used at Google for many different purposes. We attribute this success to several reasons. First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as Map Reduce computations. For example, Map Reduce is used for the generation of data for Google's production web search service, for sorting, for data mining, for machine learning, and many other systems. Third, we have developed an implementation of Map Reduce that scales to large clusters of machines comprising thousands of machines. The implementation makes efficient use of these machine resources and therefore is suitable for use on many of the large computational problems encountered at Google. We have learned several things from this work. First, restricting the

programming model makes it easy to parallelize and distribute computations and to make such computations fault-tolerant. Second, network bandwidth is a scarce resource. A number of optimizations in our system are therefore targeted at reducing the amount of data sent across the network: the locality optimization allows us to read data from local disks, and writing a single copy of the intermediate data to local disk saves network bandwidth. Third, redundant execution can be used to reduce the impact of slow machines, and to handle machine failures and data loss.

Mining data streams has been a focal point of research interest over the past decade. Hardware and software advances have contributed to the significance of this area of research by introducing faster than ever data generation. This rapidly generated data has been termed as *data streams*. Credit card transactions, Google searches, phone calls in a city, and many others\are typical data streams. In many important applications, it is inevitable to analyze this streaming data in real time. Traditional data mining techniques have fallen short in addressing the needs of data stream mining. Randomization, approximation, and adaptation have been used extensively in developing new techniques or adopting exiting ones to enable them to operate in a streaming environment. This paper reviews key milestones and state of the art in the data stream mining area.

Classical classification methods usually assume that pattern recognition models do not depend on the timing of the data. However, this assumption is not valid in cases where new data frequently become available. Such situations are common in practice, for example, spam filtering or fraud detection, where dependencies between feature values and class numbers are continually changing. Unfortunately, most classical machine learning methods (such as decision trees) do not take into consideration the possibility of the model changing, as a result of so-called *concept drift*, and they cannot adapt to a new classification model. This paper focuses on the problem of concept drift, which is a very important issue, especially in data mining methods that use complex structures (such as decision trees) for making decisions. We propose an algorithm that is able to co-train decision trees using a modified NGE (*Nested Generalized Exemplar*) algorithm. The potential for adaptation of the proposed algorithm and the quality thereof are evaluated through computer experiments, carried out on benchmark datasets from the UCI Machine Learning Repository.

A modification of the NGE algorithm has been presented in this paper. This modification could be interpreted as an information fusion model, because in producing a knowledge representation in the form of hyper rectangles, it uses rules and a training set simultaneously. The proposed algorithm was evaluated through computer experiments. The results obtained are encouraging, enabling us to continue working on algorithms from the NGE family. It is worth noting that classifiers obtained from algorithms based on the NGE concept are easily adaptable because if new training objects become available, we can improve the hyper rectangles by starting an NGE procedure for the new objects. This isnot commonly the case in classifiers trained by machine learning methods, e.g., we cannot improve given decision trees by learning new objects. A second advantage of using the NGE method is the low computational cost for classification. Additionally, the quality of the nonnested NGE makes it worthwhile to start working on an implementation thereof as a method for decision tree co-training. The results obtained are promising and as such, we intend to carry out new experiments on a wider range of datasets.

The nearest or near-neighbor query problems arise in a large variety of database applications,usually in the context of similarity searching. Oflate, there has been increasing interest in buildingsearch/index structures for performing similaritysearch over high-dimensional data, e.g., imagedatabases, document collections, time-seriesdatabases, and genome databases. Unfortunately, all known techniques for solving this problem fallprey to the \curse of dimensionality." That is,the data structures scale poorly with data dimensionality;in fact, if the number of dimensionsexceeds 10 to 20, searching in k-d trees and relatedstructures involves the inspection of a largefraction of the database, thereby doing no betterthan brute-force linear search. It has been suggestedthat since the selection of features and the choice of a distance metric in typical applicationsis rather heuristic, determining an approximatenearest neighbor should suffice for most practicalpurposes. In this paper, we examine a novelscheme for approximate similarity search basedon hashing. The basic idea is to hash the points from the database so as to ensure that the probability of collision is much higher for objects that are close to each other than for those that are far apart. We provide experimental evidence that our method gives significant improvement in running time over other methods for searching in high dimensional spaces based on hierarchical tree decomposition.

Existing parallel mining algorithms for frequent item sets lack a mechanism that enables automatic parallelization, load balancing, data distribution, and fault tolerance on large clusters. As a solution to this problem, we design a parallel frequent item sets mining algorithm Map Reduce programming model. To achieve compressed storage and avoid building conditional pattern bases, incorporates the frequent items ultra-metric tree, rather than conventional FP trees. In Map Reduce jobs are implemented to complete the mining task. In the crucial third Map Reduce job, the mappers independently decompose item sets, the reducers perform combination operations by constructing small ultra-metric trees, and the actual mining of these trees separately. We implement an in-house cluster. We show that the cluster is sensitive to data distribution and dimensions, because item sets with different lengths have different decomposition and construction costs. To improve the performance, we develop a workload balance metric to measure load balance across the cluster's computing nodes. We developed an extension, to speed up the mining performance for high-dimensional data analysis. Extensive experiments using real-world celestial spectral data demonstrate that our proposed solution is efficient and scalable

In many applications of information systems learning algorithms have to act in dynamic environments where data are collected in the form of transient data streams. Compared to static data mining, processing streams imposes new computational requirements for algorithms to incrementally process incoming examples while using limited memory and time. Furthermore, due to the non-stationary characteristics of streaming data, prediction models are often also required to adapt to concept drifts.

Out of several new proposed stream algorithms, ensembles play an important role, in particular for no stationary environments. This paper surveys research on ensembles for data stream classification as well as regression tasks. Besides presenting a comprehensive spectrum of ensemble approaches for data streams, we also discuss advanced learning concepts such as imbalanced data streams, novelty detection, active and semi-supervised learning, complex data representations and structured outputs.

The requirements to processing Big Data and privacy is when dealing with massive data streams, algorithms should be able to handle not only changing data, but also big volumes of instances arriving rapidly. At the same time, an ensemble for such data must still work under strict time and memory constraints. This can be handled in two ways {by proposing algorithms with improved scalability or by using special performance computing environments, like SPARK, Hadoop or GPU clusters. Although some attempts to extend the most often used software, like MOA, have already been undertaken, there is still a need for efficient implementations of existing methods within these specialized frameworks for Big Data, as well as developing new solutions natively for them. Another aspect of analyzing Big Data concerns the requirements for privacy protection, especially in complex systems where streams are a sub-part of a more complex analytical workflow. Here, often not only no information can be leaked outside, but also the teams participating within the analysis may not be willing to directly share their data. It raises the need for data stream ensemble algorithms able to work in such scenarios without the possibility of reverse-engineering the underlying data from their decisions and models.

Ensemble-based methods are among the most widely used techniques for data stream classification. Their popularity is attributable to their good performance in comparison to strong single learners while being relatively easy to deploy in real world applications. Ensemble algorithms are especially useful for data stream learning as they can be integrated with drift detection algorithms and incorporate dynamic updates, such as selective removal or addition of classifiers. Here we proposes for data stream ensemble learning as derived from reviewing over sixty algorithms. Important aspects such as combination, diversity and dynamic updates, are thoroughly discussed. Additional contributions include a listing of popular open source tools and a discussion about current data stream research challenges and how they relate to ensemble learning (big data streams, concept evolution, feature drifts, temporal dependencies, and others).

The amount of data generated by smart phones, social networks and all kinds of sensors has grown tremendously. All these data are only useful if efficiently processed so that individuals can make timely decisions based on them. Recently, a lot of progress has been made towards obtaining useful models out of massive amounts of rapidly generated data under the research area of data stream mining. Data streams pose several challenges for learning algorithms, including, but not limited to: concept drifts, temporal dependencies massive amount of instances, limited labeled instances, novel classes, feature drifts and restricted resources (time and memory) requirements. On top of that, problems found in a batch learning setting are also present in a data stream context, e.g., absent values, over fitting, noise, irrelevant features, class imbalance, and others.

In recent years, ensembles of learners have been widely studied and deployed in real world problems. They provided three reasons that justify using ensembles instead of single learners, i.e., statistical, computational and representational. Another explanation for this preference is the difficulty of obtaining a strong learner, while a set of weak learners is relatively easy to develop and can effectively be boosted into a strong learner, as long as they are trained and combined strategically. Ensemble learners are popular in the data stream setting because, besides leveraging weak learners, they can be used to handle general machine learning problems as well as data stream specific challenges. For instance, ensemble learners have been applied to address drifting concepts

It addresses the question of how to predict fine particulate matter given a combination of weather conditions. A compilation of several years of meteorological data in the city of Quito, Ecuador, are used to build models using a machine learning approach. The study presents a decision tree algorithm that learns to classify the concentrations of fineaerosols, into two categories (>15µg/m3 vs. <15µg/m3), from a limited number of parameters such as the level of precipitation and the wind speed and direction. Requiring few rules, the resulting models are able to infer the concentration outcome with significant accuracy. This fundamental research intends to be a preliminary step in the development of a web-based platform and smartphone app to alert the inhabitants of Ecuador's capital about the risk to human health, with potential future application in other urban areas.

Due to the fact that the assessment of air pollution in cities is a fundamental problem in terms of public health, this study proposed to look for a model to predict the concentration of fine particulate matter (PM2.5) from meteorological data in the metropolitan area of Quito, Ecuador. Six years of weather and aerosol concentration data enabled us to elaborate predictive statistical models based on machine learning for two strategic areas in the city: Cotocollao, which monitoring data shows to be a less polluted area, and Carapungo, which is one of the most contaminated areas in Quito. Although these two neighborhoods are relatively close each other, the statistical data reveals a distinct profile in regards to the effect of weather conditions (precipitation and wind components) on the PM2.5 concentrations. These differences are explained by Quito's unique landscape and the location of the principal, heavily trafficked roads. We were able to build two different models for each site based on the J48 Decision Tree algorithm, using the machine learning workbench "Weka". The two trees are quite short and, consequently, very easy to read. For both neighborhoods, the prediction of PM2.5 concentrations with a classification based on a threshold of 15µg/m3, can be summarized using only tworules. Overall, the rate of correctly classified instances is relatively high (above 65%), considering i) the complexity to forecast pollution from meteorological data in this region and ii) the challenge inherent in producing a model with a minimum of parameters and maximum accuracy. Moreover, it is important to point out that the models take into account

data produced over a long period of time in order to get the most generalizable result. Of course, as is common with machine learning, the higher the generalization, the lower the percentage of accurate classifications.

The analysis of meteorological and air quality data from two monitoring sites within Quito's complex urban landscape presented us with the opportunity to validate the use of a simple machine learning technique for forecasting fine particulate pollution. There is potential for future work involving other models that take into account a wider spatial data analysis, including all monitoring data from the last decade. In addition to the analysis presented in this work, the different models will be tested in order to compare their respective performance of prediction regarding the recordingsof air pollution for a period of up to one week, important given that the current best performance of local weather forecast - using WRF and MM5 weather forecast models - is 24 hours. A wider spatial analysis would offer us an opportunity to evaluate the model performance for more sites within the city, especially interesting due to Quito's complex terrain. Finally, after completing fundamental research to produce a fully reliable model for the region, a mobile phone application and a web-based platform could be developed to support a health alert system to warn sensitive urban populations about risks from air pollution.

Big Data concern large-volume, complex, growing data sets with multiple, autonomous sources. With the fast development of networking, data storage, and the data collection capacity, Big Data are now rapidly expanding in all science and engineering domains, including physical, biological and biochemical sciences. Here we are characterizing the features of Big data revolution and proposes a Big data processing model, from the data mining perspective. This data-driven model involves demand driven aggregation of information sources, mining and analysis, user interest modeling, and security and privacy considerations. We also analyzing the challenges in the data-driven model and also in the Big Data revolution.

## III. RELATED WORK

### 3.1INSTANCE SELECTION FRAMEWORK FOR NEAREST NEIGHBOR STREAM MINING:

Here, we present a lazy learning solution for massive and high-speed data streams. The proposed algorithm (DS-RNGE) consists of a distributed case-base and an instance selection method that enhances its performance and effectiveness.The case-base is structured using a distributed metric tree, which is entirely maintained in memory to expedite further neighbor queries.

DS-RNGE proceeds in two phases for each newly arrived batch of data.
1. An edition/update phase aimed at maintaining and enhancing the case-base.
2. A prediction phase that classifies new unlabeled data.

Both phases require fast neighbor queries to accomplish their aims. To deal with this problem, we propose a smart partitioning of the input space in which each sub tree queries only a single space partition. This scheme will allow us to parallelize the querying process across the cluster. A single top-level tree is maintained in the master node to route the elements in the first levels, where the partitioning is still coarse-grained. For each instance, the nearest element in the leaves of the top-tree is returned. The correspondence between leaf nodes and sub trees determines the local tree where the query will be performed.

As mentioned the hybrid spill trees use backtracking and redundancy to deal with classification in borders. This implies a cost in time and memory that is far from acceptable in streaming applications. The idea is to allow classification errors near borders in order to increase the computational efficiency. When the number of elements is much greater than the number of partitions, the number of instances with neighbors in a different partition and the classification error derived from this phenomenon become negligible.

Defeatist search is used as reference for our model because of its outstanding performance. Since an ever-growing and noisy case-base is unacceptable, here we uses a custom-designed instance selection method.

A improved local version of RNGE has been applied to control the insertion and removal of noisy instances. The original method has been redesigned for incremental learning from data streams. For each incoming example, a relative graph is built around the instance and a subset of its neighbors. The local graphs are then used to edit the case-base by deciding what instances should be inserted, removed or left untouched. As every step in this process is performed locally, the communication overhead is negligible.

DS-RNGE manages the following parameters.
1) *nt:* Number of sub trees and number of leaf nodes in the top-tree.
2) *ks:* Number of neighbors used to build the local graphs (instance selection phase).
3) *kp:* Number of neighbors used in the prediction phase.
4) *ro:* Indicates whether removal of examples should be performed or not.

3.2. INITIAL PARTITIONING PROCESS:

The first step is to consists of building a distributed metric tree formed by a top-tree (in the master machine) and a set of local trees (in the slave machines). This distributed tree will be queried and updated during next iterations with incoming batches of data.

From the first batch take a sample of *nt* instances to build the main tree. The sampled data should be small enough to fit in a single machine and should maximize the separability between examples to avoid overlapping in the future sub trees. The *nt* parameter is normally set to a value equal to the number of cores in the cluster. By doing this the algorithm is able to fully exploit the maximum level of parallelism in any stage. The routing tree is created where upper and lower bounds are defined to control the size of nodes. Once the top-tree is initialized, it is replicated to each machine and one subtree per leaf node is created in the slave machine. Then, every element in the first batch is inserted in the sub trees by following these steps.

1. For each element, the algorithm searches the nearest leaf node in the top-tree. According to the correspondence between leaf nodes and subtrees we can determine to which subtree each element will be sent. This process is performed in a Map phase.
2. The elements are shuffled to the subtrees according to their keys. Each subtree gets a list of elements to be inserted.
3. For each subtree all received elements are inserted to the tree in a local way. This process is performed in a Reduce phase.

The partitions/subtrees derived from this phase will be maintained during the complete process for reusability purposes, so that only the arriving instances will be moved across the network in each iteration. The following Algorithm1 explains this procedure in detail using a Map Reduce syntax.

**Algorithm1: Initial Partitioning Process**

1: INPUT: data, nt
2: *// data is the input dataset*
3: *// nt Number of leaf trees to be distrbuted acrost the nodes*
4: *sample* = smartSampling(*data*)
5: *topTree* = In the master machine, build the top M-tree using *sample* and the standard partitioning procedure explained in [41]. It will be replicated to every slave machine.
6: For each leaf node in the *topTree*, one subtree is created in a single slave machine. The resulting set of trees (stored as an RDD) is partitioned and cached for further processing.
7: **mapReduce** *e* ∈*data*
8: Find the nearest leaf node to *e* in *topTree*, and outputs a tuple with the tree's ID (key) and *e* (value). (MAP)
9: The tuple is sent to the correspondent partition and attached to the subtree according to its key. (SHUFFLE)
10: Combine all the elements with the same key (tree ID) by inserting them into the local tree. (REDUCE)
11: Return the updated tree.
12: **end mapReduce**

3.3 UPDATING PROCESS WITH EDITION:

When a new batch of data arrives, we need to start the updating process with edition. This is aimed at inserting new instances, as well as removing those that became redundant over time. At first, the algorithm computes which subtree each element falls into, following the same process described in the previous section. Once all instances are shuffled to sub trees, a local nearest neighbor search for each element is started in corresponding sub trees.

After obtaining neighbors the instance selector creates groups, where each one is formed by a new element and its neighbors. Then, local RNGE is applied on each group (as in Algorithm3). Here is to build a local graph around each group and through this graph to decide what kind of action to perform on each element (insertion, removal, or none).

Since each graph has only a narrow view of the case-base, the set of neighbors that can be removed has been limited to those that share an edge with the new element. Removal of old examples can be controlled through the binary parameter *ro*. If activated, the removal may cause a drop in the overall accuracy but the prediction process will run faster because of the reduced size. Note that the insertion process is exact in most of cases since the new elements are in the center of the graph and a suitable number of neighbors (a default value of 10) is enough to find their edges. The number of neighbors for graph construction can be controlled through the *ks* parameter. The greater the value of *ks*, the more precise and the slower is the removal.

Once decisions for each element are taken we perform insertions and removals locally in the subtrees in the same reduce phase. Notice that by doig so the neighbor query and the editing process are both performed in the same MapReduce process, thus reducing the communication overhead. The complete editing process is described in Algorithm 2.

**Algorithm 2:Updating Process With Edition**

1: INPUT: query, ks, ro
2: *// query is the data to be queried*
3: *// ks represents the number of neighbors to use in the instance selection phase.*
4: *// ro indicates whether to remove old noisy examples or not.*
5: **mapReduce** *e* ∈*data*
6: Find the nearest leaf node to *e* in *topTree* and outputs a tuple with the tree's ID (key) and *e* (value). (MAP)
7: The tuple is sent to the correspondent subtree according to its key. (SHUFFLE)
8: *neighbors* = the standard M-tree search process is launched for each element in its local subtree in order to retrieve the *ks*-neighbors of *e*. The output will consist of a tuple with *e* (key) and a list of its *ks*-neighbors (value). (REDUCE)
9: edited = apply the local RNGE algorithm (Algorithm 3) to each tuple in *neighbors*. The output consist of the insertion/removal decision for each element.
10: **if** *ro* == *true* **then**
11: Removed old noisy instances in *edited* from the tree.
12: **end if**
13: Add new correct instances in *edited* to the tree.
14: Return the updated tree.
15: **end mapReduce**

**Algorithm 3 :Local RNGE**

1: INPUT: e, ne
2: *// e incoming example*
3: *// ne set of neighbors for e*
4: Compute the local RNGE graph using *e* and *ne* following the procedure detailed in [46].
5: Mark *e* to be added iff most of its graph neighbors agree with its class
6: **for** *en* ∈*ne* **do**
7: **if** *en* is a graph neighbor of *e* and most of *en*'s graph neighbors do not agree with its class **then**
8: Mark *en* to be removed
9: **end if**
10: **end for**

The following Fig. 2 illustrates all the steps involved in DS-RNGE for one training iteration (one batch). The first part shows how the top tree is built with two examples: *e*1 and *e*2. Once the main tree is built one sub trees per element in the leaves is created in the slave nodes. Every element is also inserted in its local subtree. In the second phase a new element *e*3 arrives at the top-tree. The top-tree routes the search to the first partition, where the element is sent to perform a neighbor search. This search will allow to decide if the element should be inserted or not.

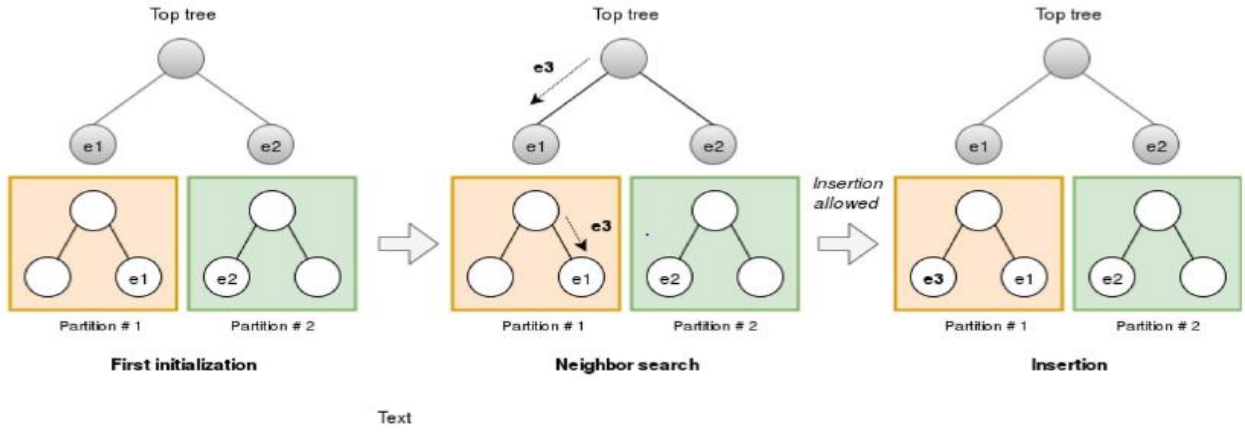


Fig2:Flow chart Describing initialization,searching,and insertion process in DS-RNGE

If the edition method decides the insertion is suitable according to its nearest neighbor (1-NN) (in this case, *e*1). The insertion is then fully local, as the element has been already sent to the correspondent node and partition. The removal process performs the same operations but removing those cases that do not agree with the edition results. In this case, the decision for *e*1 is to remain unchanged. Within the edition process, local construction of graphs and subsequent filtering is depicted in the following

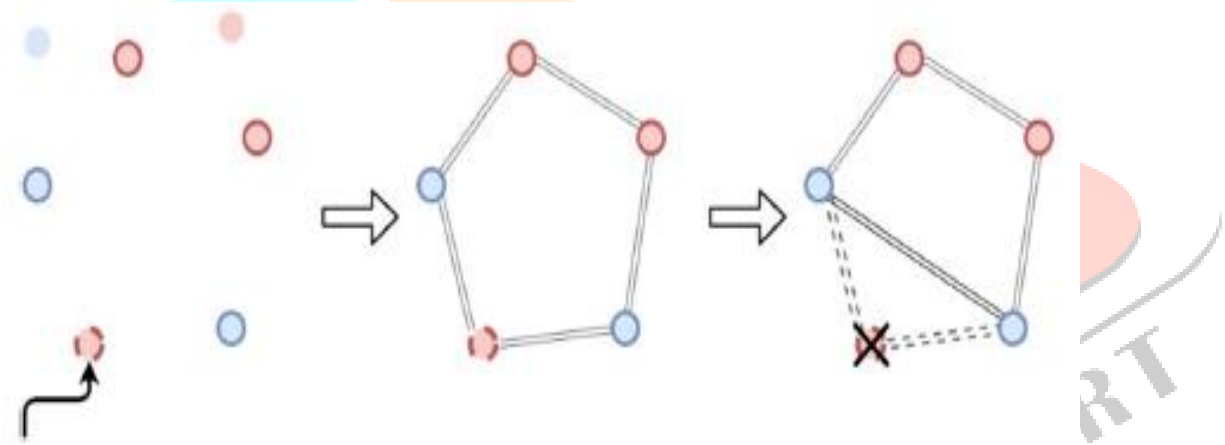

Fig3: Local graph edition for each new ex. Class A is colored in red and Class B in blue.

In this graphic a new example from class A (dashed point) arrives to a given partition (Algorithm 2). From the set of points in that partition, *ks* = 4-NN (thick circles) are selected from the pool to construct the graph shown in step 2. Note that graphs are built independently from other cases in the partition. Lastly, removal decisions are made according to the connections between neighbors. In this example, the dashed example is not inserted in the case-base since most of its edge-neighbors do not agree with its class. As there are no more agreements between nodes, no additional removals are conducted.

3.4 PREDICTION PROCESS:

Classification process is an approximate function that is started when new unlabeled data arrive at the system (see Algorithm 4). For each element the algorithm searches for the nearest leaf node in the master node and shuffles the elements to the slave machines. Next, the standard M-tree search process is used to retrieve the *kp*-neighbors of each new element. For each group, formed by a new element and its neighbors, the algorithm predicts the element's class by applying the majority voting scheme to its neighbors. Notice that the query and the prediction are both performed in the same Map Reduce phase as in the edition process.

**Algorithm 4: Prediction Process**

1: INPUT: query, kp
2: *// query is the data to be queried*
3: *// kp represents the number of neighbors for predictions.*
4: **mapReduce** *e* ∈*data*
5: Find the nearest leaf node to *e* in *topTree* and outputs
a tuple with the tree's ID (key) and *e* (value). (MAP)
6: The tuple is sent to the correspondent subtree according
to its key. (SHUFFLE)

7: *neighbors* = the standard M-tree search process is launched for each element in its local subtree in order to retrieve the *ks*-neighbors of *e*. The output will consist of a tuple with *e* (key) and a list of its *ks*-neighbors (value).
(REDUCE)
8: For each tuple in *neighbors* return the most-voted class from the list of neighbors. This value will be the class predicted for the given element.
9: **end mapReduce**

## IV. CONCLUSION

In this paper, we have presented DS-RNGE, a nearestneighbour classification solution for processing massive and high-speed data streams using Apache Spark. Up to our knowledge, the first lazy learning solution designed for large-scale, high-speed, and streaming problems. Our model organizes the instances by using a distributed metric tree, consisting of a top-level tree that routes the queries to the leaf nodes and a set of distributed subtrees that performs the searches in parallel. DS-RNGE includes an instance selection technique that constantly improves the performance and effectiveness of the learner by only allowing the insertion of correct examples and removing outdated ones. As all phases in DS-RNGE perform the computations locally, our system is able to quickly respond to the continuous stream of data. The experimental analysis shows that DS-RNGE combineshigh accuracy with significantly reduced processing time and memory consumption. This allows for an resource-efficient mining of massive dynamic data collections. DS-RNGE without removal overcomes in terms of precision the base model without edition in any case. DS-RNGE yields better time results in the prediction phase, whereas its competitor performs faster in updating the case-base. In general, both algorithms have similar performance, if we measure the total time spent in both phases. Our future work will concentrate on adding a condensation technique in order to control the ever-growing size of the case-base over time. By removing redundancy, the time cost derived from edition will be alleviated, while at the same time maintaining the original effectiveness. Additionally, we plan extend our approach to drifting data streams and propose time and memory efficient solutions for rebuilding the model as soon as the change occurs. We plan to tackle this challengeby extending our model with drift detection module, aswell as by using instance weighting with forgetting to allow for smooth adaptation to changes. Additionally, we envision modifications of our algorithm that will make it suitable for mining massive and imbalanced data streams.

## REFERENCES

**[1]**D. Han, C. G. Giraud-Carrier, and S. Li, "Efficient mining of high-speed uncertain data streams," *Appl. Intell.*, vol. 43, no. 4, pp. 773–785, 2015.
**[2]** X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data,"*IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.
**[3]** U. Fayyad and R. Uthurusamy, "Evolving data into mining solutions for insights," *Commun. ACM*, vol. 45, no. 8, pp. 28–31, Aug. 2002. [Online]. Available: http://doi.acm.org/10.1145/545151.545174
**[4]** A. Fernández *et al.*, "Big data with cloud computing: An insighton the computingenvironment, map reduce, and programming frameworks,"*Wiley Interdiscipl. Rev. Data Min. Knowl. Disc.*, vol. 4, no. 5, pp. 380–409, 2014.
**[5]**J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. OSDI*, San Francisco, CA, USA, 2004, pp. 137–150.
**[6]**A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. 25th Int. Conf. Very LargeDataBases (VLDB)*, Edinburgh, U.K., 1999, pp. 518–529.
**[7]**Y. Xun, J. Zhang, and X. Qin, "FiDoop: Parallel mining of frequent itemsets using MapReduce," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 3, pp. 313–325, Mar. 2016.
**[8]**A. Bifet, G. D. F. Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proc. 21thACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, Sydney, NSW,Australia, 2015, pp. 59–68.