Real-time scalable parallel data stream classification

Master Degree in Computer Science & Engineering

By

Roberto Dean Robledo McClymont

Supervised by: Ivan Rodero Castro Ph.D.

Department High-Performance Computing
Open University of Catalonia
June 2018

To my beloved wife, Júlia and my daughter Alba

Look up at the stars and not down at your feet. Try to make sense of what you see, and wonder about what makes the universe exist. Be curious. And however difficult life may seem, there is always something you can do and succeed at.
It matters that you don't just give up.

Stephen Hawking

Abstract

Nowadays, big data is a term heard and seen everywhere and it is due to the new digital era that we are experiencing Data sets grow rapidly because they are increasingly gathered by cheap and numerous information-sensing devices of different sizes such as mobile devices, satellites (remote sensing), software logs, cameras, microphones, radio-frequency identification, etc.
Therefore, big data contains voluminous and complex data sets that traditional data processing application software cannot deal with. Within big data there are several challenges when capturing the data, storing the data, analyzing the data, sharing, transferring, visualizing, querying and updating and even information privacy.
The five main aspects that describe what the scope of Big data is: data generated at a fast rate (Velocity), very large unknown data quantities (Volume), uncertainty in the data (Veracity) and different forms of data such as structured data, images, etc. (Variety) and the last one added is (Value) which refers to potential insights that can be derived by analyzing the data.

The main objective of this final master project is to create a real-time prototype that is capable of classifying real-time data using several deep learning algorithms. Classifying means to give "valuable" information – that maybe can be unknown - to the different incoming data. Note also that this could be extrapolated to other fields.
In addition, some research will be done in the field of deep learning with the aim of giving some guidelines about how big data can be classified in a cluster environment. The idea of developing this prototype is to prove that large amounts of data processing can be tackled within this methodology.
Further work can be done following this line with the purpose of creating a real data-time analysis methodology that can be applicable to other fields such us medical studies, economic statistics, mobility solutions and many others.
As in all research studies, iterative processing must be done in order to enhance and/or update the deep algorithms that will be presented during this final master project.

Acknowledgements

These lines are devoted to these people who made things easier for me and contributed to complete this work.
First to all, I would like to give a special mention to those who are no longer with us and taught and helped me to grow and become the person I am: Papá, Abue and Tio.
Second, thanks to my beloved Júlia for supporting me and being by my side during all this journey, for her endless patience with my long hours on the computer. To the best mother, Mum, for being always there and for her unconditional support in my entire life. To the best sister and friend Ia, thanks for your advice. To my brother-in-law and friend Dani, you always find a sense of humour in all situations.
Thanks also to my father-in-law, Oriol, and my mother-in-law, Lisi for their encouragement.
Last but not least, I would like to also thank my best friends for their support and encouragement during my master's degree: Isaac, Alex, Fede, Pey, Fran, Carles, Carlos, Gili, Eli, Vane, Tania, Eva, Kadu, Jose, Gabri, Ruben, Alex L., Pablo.

Table of Contents

List of Figures

List of Tables

1. Introduction

This final Master project tackles how big data from several interdisciplinary observatories can be classified. To do so, multiple open source technologies are used to handle real-time data stream processing.

1.2 Technologies

1.2.1 Apache Storm

Regarding the data processing, the architecture is centered on the use of the Apache storm which is a free and open source distributed real-time computation system. Storm can process millions of tuples per second per node. It is scalable, fault-tolerant and guarantees that your data will be processed and is easy to set up and operate.
A storm cluster is similar to a Hadoop cluster. Whereas in Hadoop you run "MapReduce jobs", in storm you run "topologies". One key difference is that MapReduce jobs eventually finish, whereas a topology processes messages forever unless you kill it.
The Apache Storm framework follows the master-slave paradigm, sometimes it is called a farm of processes. This paradigm refers to the fact that the master process is responsible for assigning jobs to other processes "slaves". There are three kinds of nodes in a Storm cluster: master, worker and Zookeeper nodes. Master node runs a daemon called "Nimbus" and it is responsible for distributing codes around the cluster which means assigning tasks to nodes and monitoring failures. On the other hand, a worker runs a daemon called "Supervisor" and listens for work assigned to its machine and starts and stops workers processes as necessary based on what the master (Nimbus) has assigned to it.
In order to do real-time computation in Storm, you must create what are called "topologies". A topology is a graph computation which contains the processing logic and links between nodes to indicate how data should be distributed around the nodes. When a topology is "run", it means that each worker node process a subset of a designed topology with the idea to distribute the amount of work in the nodes to obtain a horizontal scalability. Last but not least, the Zookeeper node is responsible for coordinating the Nimbus (master) and the Supervisors. All states are kept in Zookeeper or in a local disk which means that both daemons (nimbus and supervisors) are fail-fast and stateless. This design leads to Storm clusters being extremely stable.

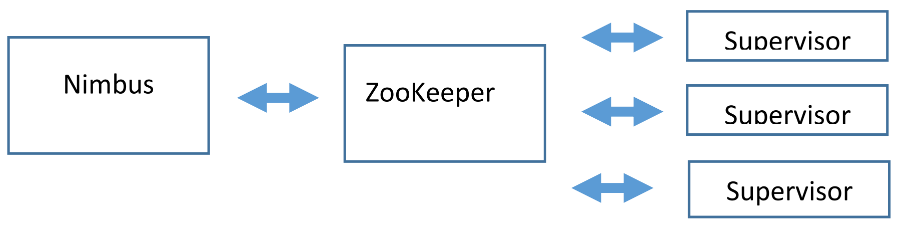


Figure 1: Nodes Interaction between the Nimbus, Supervisor and Zookeeper

1.2.2 Apache Kafka

Apart from the Apache Storm technology for data processing described previously, it is also important to define a message system and this will be Kafka which is used for building real-time data pipelines and streaming applications. It is also horizontally scalable and fault-tolerant. Moreover, it has low latency and high throughput message handling. It is also configurable to have its internal data streams which are called "Topics" distributed in a parallel way as required. A topic is a category or feed name in which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.
These published topics persist, even if they have not been consumed using a configurable retention period.

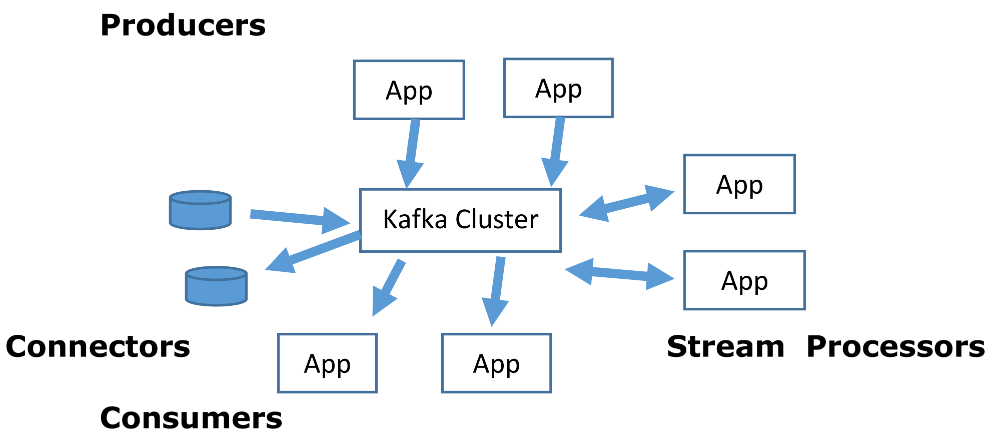


Figure 2: Kafka communication between clients and servers

Note also that the Zookeeper node is used as a distributed coordinator and a topic consumer offset manager by Kafka.

1.3 High Level Main Flow diagram

The main flow of our prototype can be described with the following steps:
First, as we already commented, we will receive real-time data from different interdisciplinary data observatories (different domains). As can be seen in the following figure 3, data will be pre-processed (if needed) which basically means that some useless/unused data will be filtered. When the pre-processing is done data will be pushed into a Kafka Topic. Meanwhile, Spout nodes will be listening to one or several Kafka topics and they will emit any tuple to the bolt nodes in which they can do filtering, functions, aggregations, joins, talking to the database, but in our case the main function will be applying the different deep learning algorithms in order to produce a value product that will be emitted to the output Kafka output stream and can be used as information (historical data) for the following incoming data.

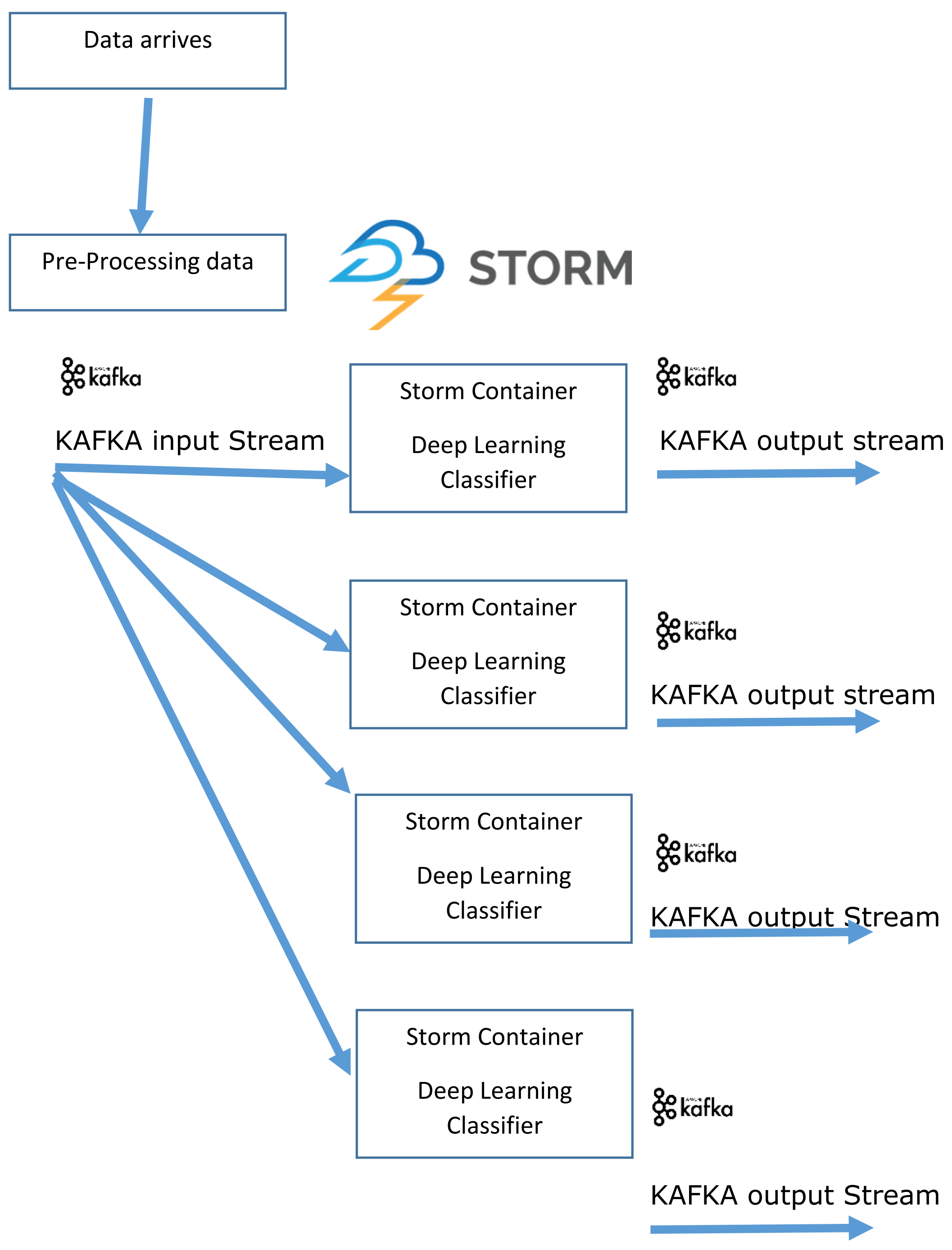
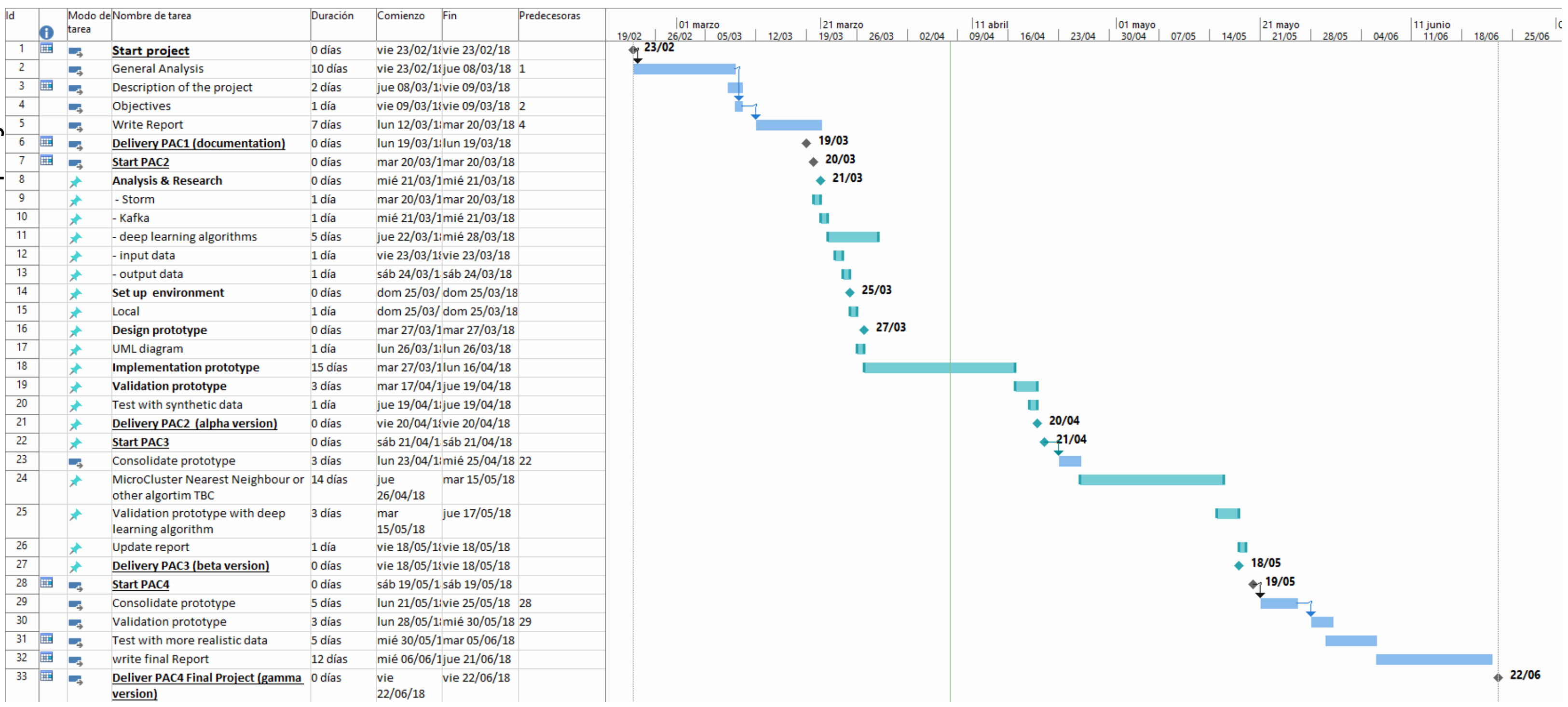


Figure 3: Architecture of Computer nodes for a real-time data classification

1.4 Schedule master’s final project

A Gantt chart is used to illustrate the final master project schedule



Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Start project**	0 días	vie 23/02/1	vie 23/02/18	
2		General Analysis	10 días	vie 23/02/1	jue 08/03/18	1
3		Description of the project	2 días	jue 08/03/1	vie 09/03/18	
4		Objectives	1 día	vie 09/03/1	vie 09/03/18	2
5		Write Report	7 días	lun 12/03/1	mar 20/03/18	4
6		**Delivery PAC1 (documentation)**	0 días	lun 19/03/1	lun 19/03/18	
7		**Start PAC2**	0 días	mar 20/03/1	mar 20/03/18	
8		**Analysis & Research**	0 días	mié 21/03/1	mié 21/03/18	
9		- Storm	1 día	mar 20/03/1	mar 20/03/18	
10		- Kafka	1 día	mié 21/03/1	mié 21/03/18	
11		- deep learning algorithms	5 días	jue 22/03/1	mié 28/03/18	
12		- input data	1 día	vie 23/03/1	vie 23/03/18	
13		- output data	1 día	sáb 24/03/1	sáb 24/03/18	
14		**Set up environment**	0 días	dom 25/03/	dom 25/03/18	
15		Local	1 día	dom 25/03/	dom 25/03/18	
16		**Design prototype**	0 días	mar 27/03/1	mar 27/03/18	
17		UML diagram	1 día	lun 26/03/1	lun 26/03/18	
18		**Implementation prototype**	15 días	mar 27/03/1	lun 16/04/18	
19		**Validation prototype**	3 días	mar 17/04/1	jue 19/04/18	
20		Test with synthetic data	1 día	jue 19/04/1	jue 19/04/18	
21		**Delivery PAC2 (alpha version)**	0 días	vie 20/04/1	vie 20/04/18	
22		**Start PAC3**	0 días	sáb 21/04/1	sáb 21/04/18	
23		Consolidate prototype	3 días	lun 23/04/1	mié 25/04/18	22
24		MicroCluster Nearest Neighbour or other algortim TBC	14 días	jue 26/04/18	mar 15/05/18	
25		Validation prototype with deep learning algorithm	3 días	mar 15/05/18	jue 17/05/18	
26		Update report	1 día	vie 18/05/1	vie 18/05/18	
27		**Delivery PAC3 (beta version)**	0 días	vie 18/05/1	vie 18/05/18	
28		**Start PAC4**	0 días	sáb 19/05/1	sáb 19/05/18	
29		Consolidate prototype	5 días	lun 21/05/1	vie 25/05/18	28
30		Validation prototype	3 días	lun 28/05/1	mié 30/05/18	29
31		Test with more realistic data	5 días	mié 30/05/1	mar 05/06/18	
32		write final Report	12 días	mié 06/06/1	jue 21/06/18	
33		**Deliver PAC4 Final Project (gamma version)**	0 días	vie 22/06/18	vie 22/06/18	

Figure 4: Schedule Final Project

1.5 Environment

In this section, all the environment used within the rtDeep prototype will described.

1.5.1 Programming language

Object-Oriented programming (OOP) is a programming paradigm based on the concept of "objects". These objects contain data which are called attributes and also "behavior" in the form of procedures/methods. In OOP, computer programs are designed by making them out of objects that interact. OOP languages are class-based, meaning that objects are instances of classes. Therefore, Java programming language will be used to implement the prototype taking into account that the Apache storm is compatible with Java.

1.5.2 Eclipse

In order to facilitate the implementation of the real-time prototype, an integrated development environment (IDE) will be used and this will be the Eclipse Neon.

1.5.3 Maven Compilation

The Apache Maven will be used as an automatic building tool which is used primarily for Java projects. Maven addresses two aspects of building software:
First it describes how software is built and second it describes its dependencies. To do so, an XML file is used to describe the software project being built, its dependencies on other external modules and components, the build order, directories and required plug-ins. Note also that Maven dynamically downloads Java libraries and Maven plug-in from one or more repositories.

1.5.4 Pre-requirements (Zookeeper and Kafka Server)

Before using Storm two pre-requirements must be taken into account.
Regarding the first requirement, as we already pointed out nimbus and supervisor states are kept in a Zookeeper server. The main role of this Zookeeper server is to coordinate a cluster. In order words, a Zookeeper provides a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. Therefore, the Apache Zookeeper daemon must be set up and launched before running Storm. The second requirement is related to the messaging system and this means that a Kafka server daemon must be set up and launched before running Storm. Storm topology will ingest tuples from different Kafka topic names and it will push tuples (output) into different topics.

1.5.5 Configuration File

There are many parameters that can be filled in within the topology and the properties file format has been chosen. A ".properties" is a file extension for files mainly used in Java related to technologies to store the configurable parameters of an application. Within a properties file, each parameter is stored as a pair of string, one storing the name of the parameter (the key) and the other storing the value of its parameter.

Comment lines in "*.properties" files are denoted by the number sign(#) as the first non-blank character, in which all the remaining text on that line is ignored.

1.5.6 Versions and dependencies

The following table contains a summary of the versions of the OS, modules, tools, etc. used during the implementation of rtDeep prototype.

Name	**Version**
Linux Ubuntu distribution	16.04
Eclipse Neon 2	4.6.2
Java SDK	1.8.0_161
Apache Storm	1.2.0
Apache Kafka	1.0.1
Apache Zookeeper	3.4.6
Python	3.6.0
SkLearn	0.18
Matplotlib	2.0.2
CSV (python module)	1.0

Table 1: Libraries with their version used in rtDeep implementation

1.6 Deliveries

As can be seen in the Gantt chart above, three different versions will be provided alpha, beta and gamma respectively.
The Alpha version will mainly contain the design and implementation of the data segmentation that will be passed within the topology. The Beta version will include storm topology and deep learning design and implementation. The Gamma version is the final version of the prototype.

2. Input/Output Data

In production mode, the topology will pull data from different Kafka topics to check if there are any new incoming tuples to ingest them directly to the topology. Nonetheless, during the implementation synthetic data are generated to fill in the topics.

This synthetic data will be created in JSON format which is an open standard file format that uses human-readable text to transmit data objects consisting of attribute-values pairs and array data types. JSON filenames use the extension .json.

Kafka lets us publish and subscribe to streams of records which can be of any type. Kafka gives users the ability to create their own serialization and deserialization. On the one hand, the serialization is the process of converting an object into a stream of bytes that are used for transmission. Then, Kafka stores and transmits these bytes of arrays in its queue. On the other hand, deserialization converts the bytes of arrays into the desired data type. Kafka by default provides serializers and deserializers for the common data types: String, Long, Double, Integer, etc.)

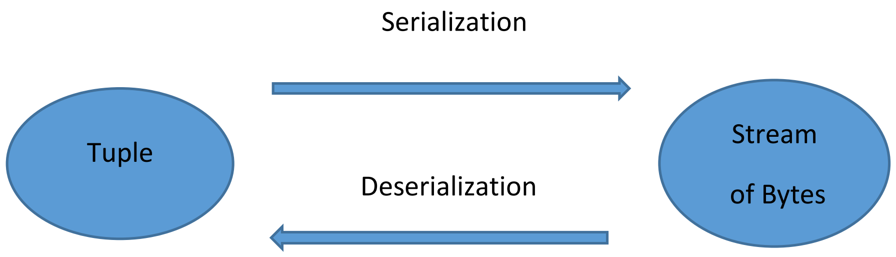


Figure 5: Serialization and Deseralization process

In order to make robust and flexible input data, tuple format will be used. Tuple is a list of pair values which has its key and value. This tuple format lets us design complex structures of data. Nonetheless, Kafka serialization and deserialization interfaces will be overwritten.

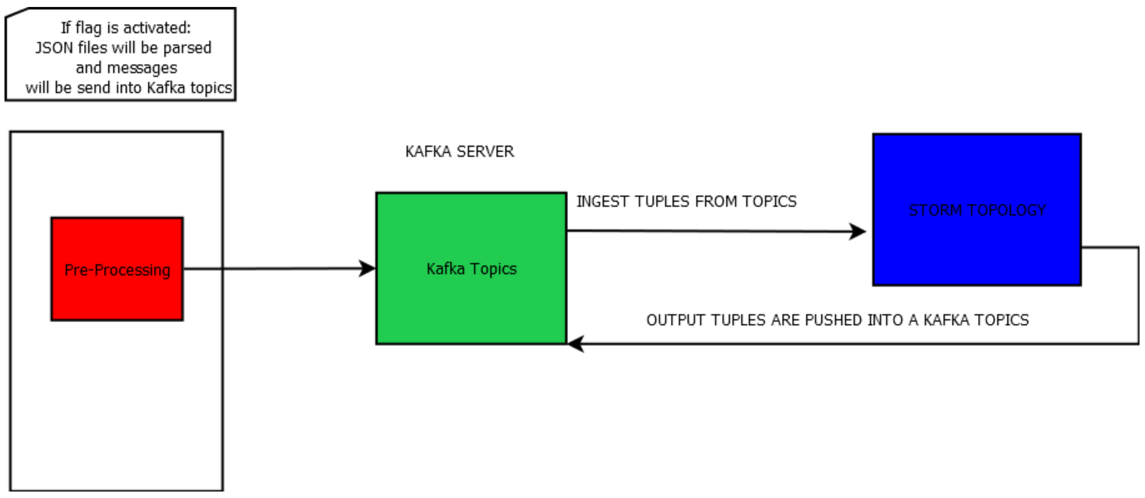

If flag is activated:
JSON files will be parsed
and messages
will be send into Kafka topics
Pre-Processing
KAFKA SERVER
Kafka Topics
INGEST TUPLES FROM TOPICS
STORM TOPOLOGY
OUTPUT TUPLES ARE PUSHED INTO A KAFKA TOPICS

Figure 6: Synthetic Input Data Diagram

As can be seen in figure 6, a flag can be enabled or disabled in a properties file before running the topology. If this flag is enabled, a pre-processing step will be run. This pre-processing will create synthetic data parsing the data from JSON files in a directory and messages are sent in the same format as the incoming real data.

As you can see, Storm topology (blue box) will expect messages from a Kafka topic. Then processing will be computed generating output tuples that will be pushed again into a new Kafka topic, meaning that the pre-processing is unknown to Storm itself.

2.1 Modelling input variability

In order to make more realistic synthetic input data two statistical distributions which are defined by some mathematical function or probability density function (PDF) will be used: normal and uniform distribution. On the one hand, normal distribution is specified by two parameters: mean (its location) and standard deviation (its spread). On the other hand, uniform distribution is specified also by two parameters: min and max value.

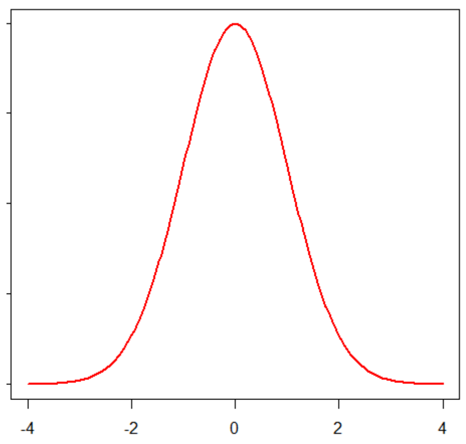


Figure 7: Normal Distribution (mean=0, std=4)

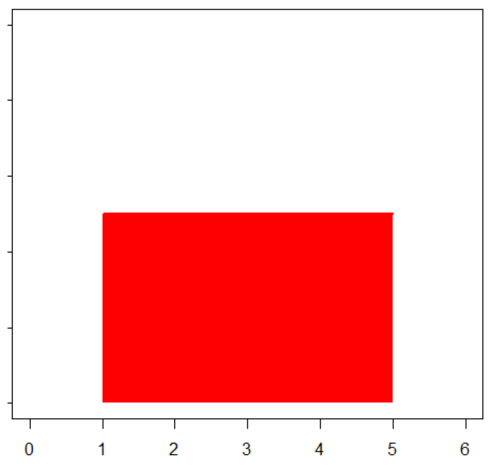
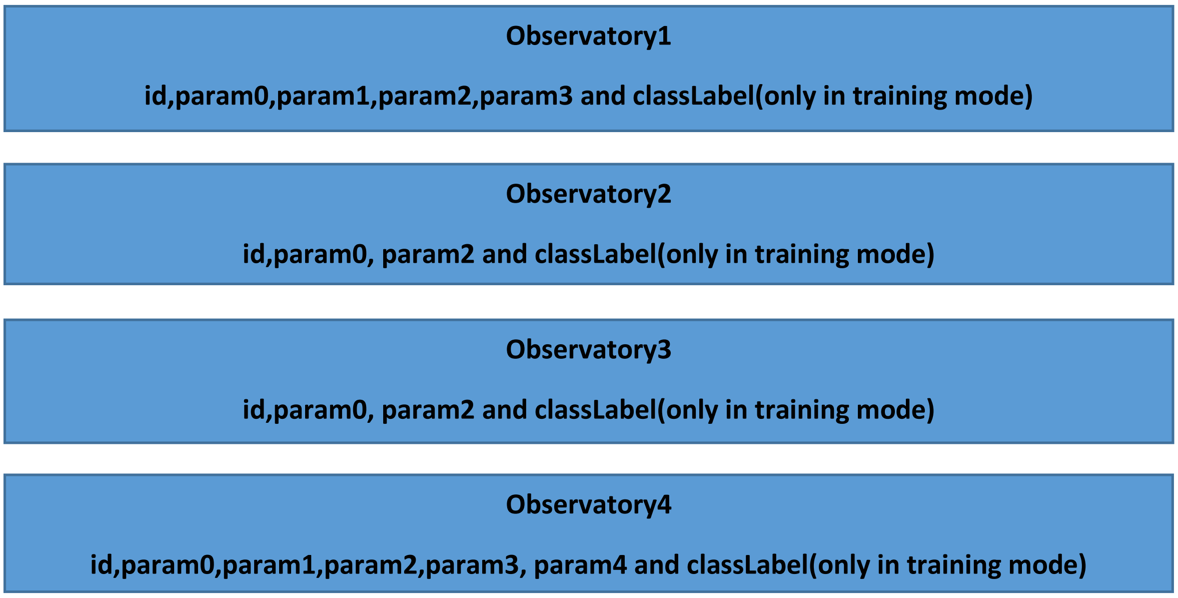


Figure 8: Uniform distribution (min=1, max=5)

Apart from using two different statistical distributions, a simulation of receiving input parameters from four different observatories which measure several interdisciplinary parameters from the sea will be implemented. Hence, two scenarios are created: a normal and tsunami case. To do so, first it is important to understand that each Observatory will contain a list of parameters with its Id, following a normal distribution for all the items of observatory 1 and 2 respectively and a uniform distribution for all the items of observatory 3 and 4. Moreover, a class label item will added for the first 'n' input tuples of each observatory that will be used in train mode.



Observatory1

id,param0,param1,param2,param3 and classLabel(only in training mode)

Observatory2

id,param0, param2 and classLabel(only in training mode)

Observatory3

id,param0, param2 and classLabel(only in training mode)

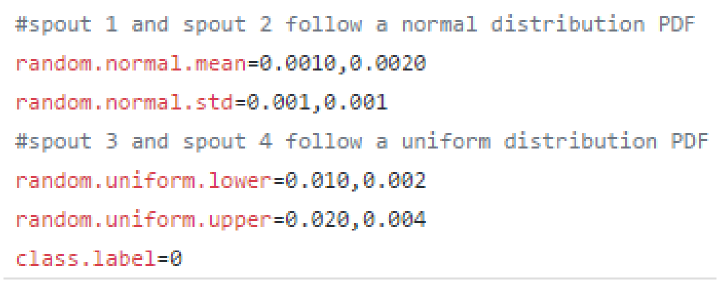
Observatory4

id,param0,param1,param2,param3, param4 and classLabel(only in training mode)

Figure 9: Simulation of the content of each observatory

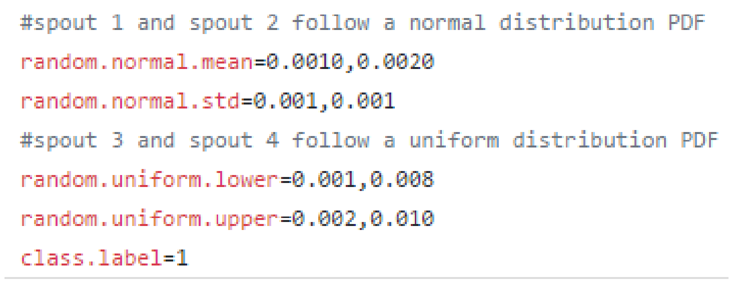
The specified parameters for the generation of the distributions will be read from a properties file.

To simulate the two different scenarios (normal o tsunami) the uniform distribution of observatory 3 and 4 values will be slightly changed. This will lets us emulate the different measurements when detecting that a tsunami is about to happen.



```
#spout 1 and spout 2 follow a normal distribution PDF
random.normal.mean=0.0010,0.0020
random.normal.std=0.001,0.001
#spout 3 and spout 4 follow a uniform distribution PDF
random.uniform.lower=0.010,0.002
random.uniform.upper=0.020,0.004
class.label=0
```

Figure 10: Properties file with uniform and normal distribution parameters for normal scenario



```
#spout 1 and spout 2 follow a normal distribution PDF
random.normal.mean=0.0010,0.0020
random.normal.std=0.001,0.001
#spout 3 and spout 4 follow a uniform distribution PDF
random.uniform.lower=0.001,0.008
random.uniform.upper=0.002,0.010
class.label=1
```

Figure 11: Properties file with uniform and normal distribution parameters for tsunami scenario

It is mandatory to check that the synthetic input data is correct and to do so, a Multidimensional scaling (MDS) algorithm is used. MDS is a means of visualizing the level of similarity of individual cases of a dataset. It refers to a set of related ordination techniques used in information visualization, in particular to displaying the information contained in a distance matrix. It is a form of non-linear dimensionality reduction. An MDS algorithm aims to place each object in N-dimensional space so that the between-object distances are preserved as well as possible. Each object is then assigned coordinates in each of the N dimensions. The number of dimensions of an MDS plot N can exceed 2 and is specified a priori.

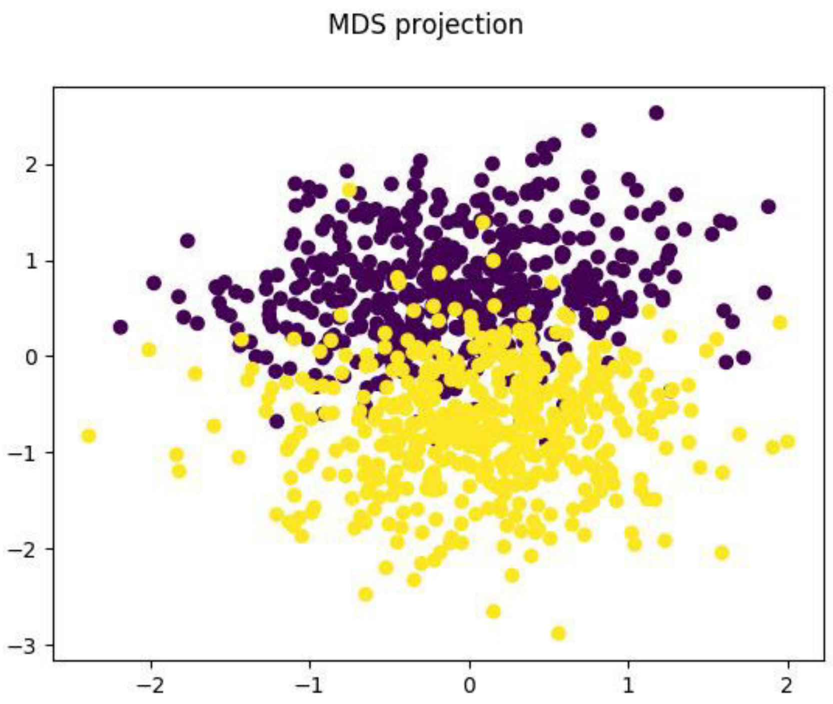


Figure 12: Plot using the MDS algorithm for the two noted scenarios (normal and tsunami)

3. Topology

In order to do real-time computation in Storm, you must create what are called "topologies". A topology is a graph computation which contains the processing logic and links between nodes to indicate how data should be distributed around the nodes. Within a topology three concepts are important: stream, spouts and bolts. The stream is the core abstraction in Storm. A stream is an unlimited sequence of tuples that is processed and created in parallel in a distributed fashion. Each stream is defined with a schema that names the fields in the stream's tuples and its type like integer, float, etc. Hence, the data will be pre-processed to fulfill the stream condition before the tuple is ingested into the topology. The other two concepts are related to the different roles that a node can have.

3.1 Spout

Spout is a source of streams in a topology. Spouts are in charge of reading tuples from an external source (a Kafka cluster in our case) and they will emit them into the topology. Spouts can either be reliable or unreliable. A reliable spout is capable of replaying a tuple if it failed to be processed by Storm, whereas an unreliable spout forgets about the tuple as soon it is emitted. There are two methods within storm 'ack' and 'fail' that called only for reliable spouts.

3.2 Bolt

All processing in topologies is done in bolts. Bolts can do anything from filtering, functions aggregations, joins, talking to a database and more. In our case bolts will compute the deep learning algorithm in order to obtain "value" (one of the V's of big data) for our huge amount of data.
Bolts can do simple stream transformations. If there are complex stream transformations it is usually done in multiple steps and thus multiple bolts.
When creating bolt's input streams, you must subscribe to specific streams of another component (spout or bolt).

3.3 Join operation

Storm supports joining multiple data streams into one stream. Join bolt is a windowed bolt that waits for a configured window duration to match up the tuples among the streams being joined. A stream will be joined with the other streams using the field in which it has been FieldsGrouped. Within the merge operation, it is mandatory to specify what the output fields are in the select() method which are the output fields. The argument in the select method is a comma separated list of fields. Individual names can be prefixed with a stream to differentiate between the same fields in multiple stream. For instance:

"stream1: item1, stream2: item1"

3.4 Classifiers

Note that at the beginning of the topology execution, a set of data instances will be used to train. When a certain number of tuples are passed (training mode), topology will directly pass to Test Mode. Each classifier data instance will receive a field called "processId" with its data instance. In training mode, the classifier will only apply the MC-NN algorithm if the "processId" matches its own Id or if it is equal to "all" (TRAINING mode).

On the other hand, in TEST mode the input stream will be processed in all the classifier instances and each of them will predict a class label.

3.5 RtDeep Topology Design

One of the critical points in our prototype implementation is the design of the Storm topology. The design of the topology describes how data is segmented or replicated. Moreover, it defines the number of spouts and bolts that the topology will contain and describes the stream grouping for each component which defines how the streams are passed between the components; it defines how the stream should be partitioned among the bolt's tasks. In order words, it defines for each bolt which stream it should receive as input.

In storm there are eight built-in stream groupings:

1) Shuffle grouping: Tuples are randomly distributed across the bolt's tasks in a way that each bolt is guaranteed to get an equal number of tuples.
2) Fields grouping: The stream is portioned by the field specified in the grouping. For instance if the stream is grouped by the "Id" field, tuples with the same "Id" will go always to the same task which gives us a horizontal scalability.
3) Partial Key grouping: The stream is partitioned by the fields specified in the grouping (like Field grouping) but are load balanced between two downstream bolts.
4) All grouping: The stream is replicated across all the bolt's tasks. Use this grouping with care.
5) Global grouping: The entire stream goes to a single bolt tasks.
6) None grouping: This grouping specifies that it does not matter how the stream is grouping. It is equivalent to shuffle grouping.
7) Direct grouping: A stream grouped this way means that the producer of the tuples decides what task of the consumer will receive this tuple.
8) Local or shuffle grouping: If the target bolt has one or more tasks in the same worker process, tuples will be shuffled to only those in-process tasks. Otherwise, this acts like a normal shuffle grouping.

Before designing the topology, it is mandatory to know exactly how the data is going to be received and to do so some assumptions were made. First, taking into account that the data will come from different observatories it is assumed that each incoming data from each observatory will push the data into a different topic. Hence, with this assumption the topology must contain four spouts; each of them will be responsible for listening to a different topic that corresponds to a different observatory.

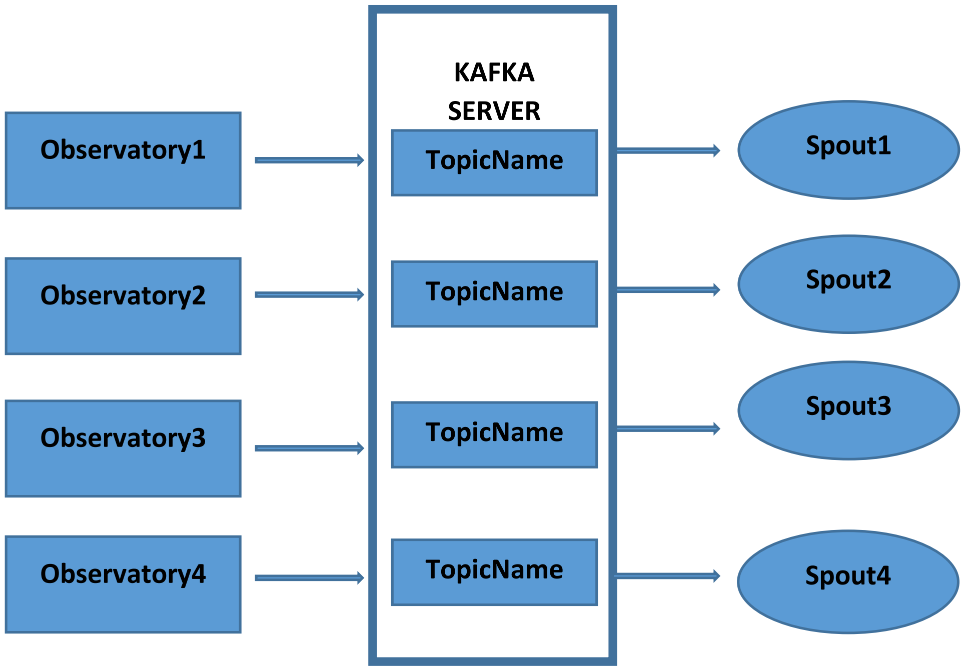


Figure 13: Each observatory will push data into a different topic name and a Spout will listen to it.

Another important assumption made is that the frequency of the data must be the same. However, if for some reason this assumption is changed pre-processing must be made in the data to homogenize data or some aggregations operators must can be added into the topology.

Last but not least, the third assumption made is that all incoming data from each observatory must contain an identifier that will be used afterwards in the topology to merge the inputs. This identifier could be for example the timestamp because all incoming data are sampled in the same frequency (second assumption) or it could be even simpler with an integer.

Once the assumptions are denoted, the topology data can be presented and analyzed in detail as can be seen in the following figures 14,15,16,17. Storm Topology will be launched and it will be waiting for new incoming data from each observatory. Therefore, four spouts are created, which are green in figure 14, and they are responsible for listening to a different Kafka topic name and they also emit the 'raw' tuples directly into the topology without any modifications.

The input tuples emitted by the spouts are sent into a FilterKafkaSpout (navy blue) which are bolts that filter any irrelevant data and also perform a stream 'field grouping' aggrupation that is needed before the Join operation (grey) and in which 'Id' parameter will be used as the join key value. The split of the stream data in several nodes, using the field grouping, allows us to take advantage of horizontal scalability when using a cluster in which each node will process different join key values.

When tuples with the same key (identifier) have arrived and passed the filter bolt, a join operation –JOIN-INPUT-STREAMS will merge them into a single tuple.

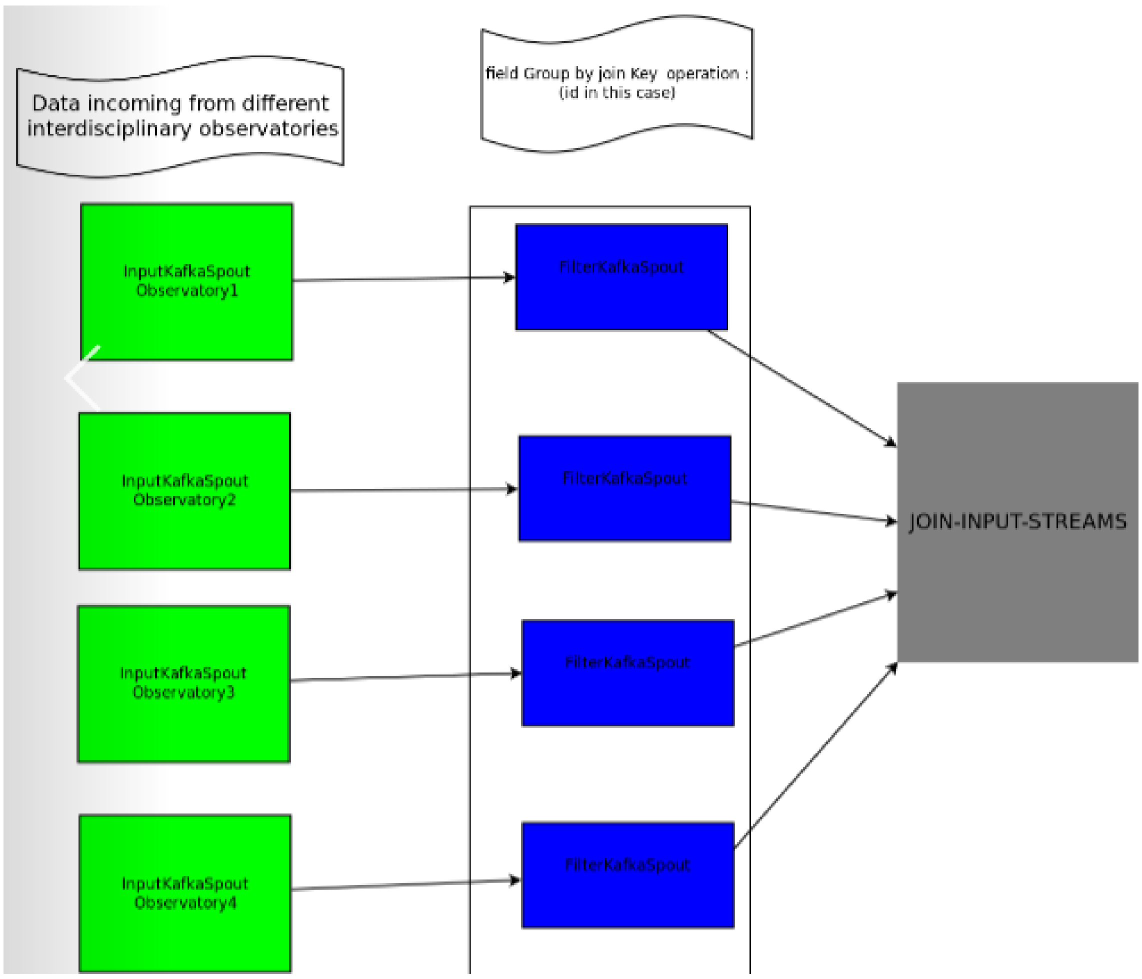

Data incoming from different interdisciplinary observatories
field Group by join Key operation : (id in this case)
InputKafkaSpout Observatory1
InputKafkaSpout Observatory2
InputKafkaSpout Observatory3
InputKafkaSpout Observatory4
FilterKafkaSpout
FilterKafkaSpout
FilterKafkaSpout
FilterKafkaSpout
JOIN-INPUT-STREAMS

Figure 14: Initial Storm rt Deep Step

When tuples are merged in the join operator (grey) a global grouping stream will be performed which means that all streams go to the same partition and this is because the following "FilterTrainTest" bolt is responsible for deciding if the tuple is processed or not by the following classifiers instances changing from training to test mode.

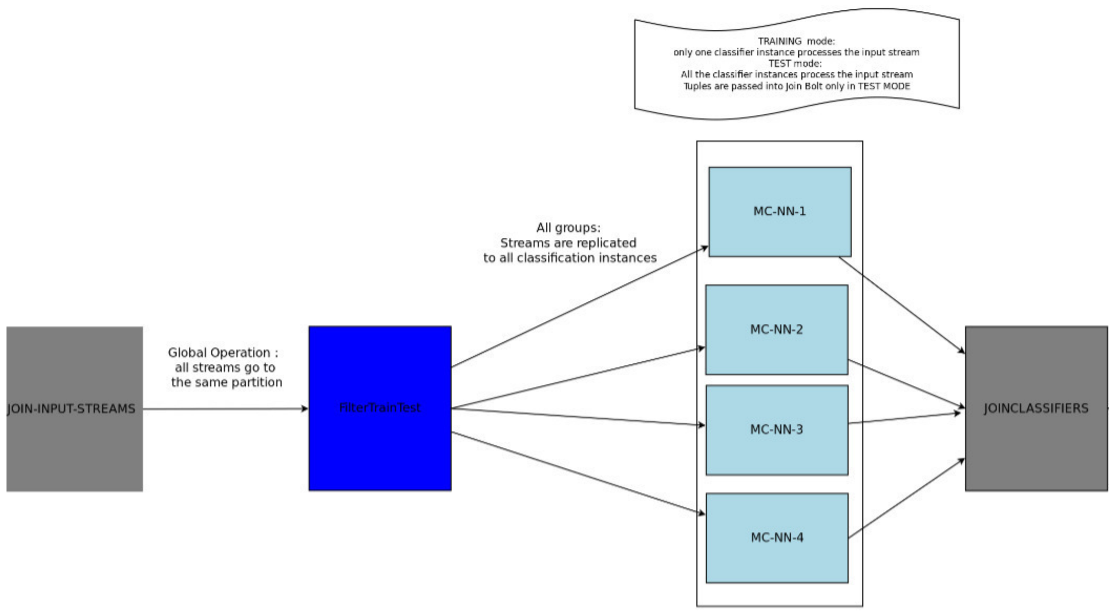

TRAINING mode:
only one classifier instance processes the input stream
TEST mode:
All the classifier instances process the input stream
Tuples are passed into Join Bolt only in TEST MODE
All groups:
Streams are replicated
to all classification instances
Global Operation :
all streams go to
the same partition
JOIN-INPUT-STREAMS
FilterTrainTest
MC-NN-1
MC-NN-2
MC-NN-3
MC-NN-4
JOINCLASSIFIERS

Figure 15: After the input tuples are merged(JOIN-INPUT-STREAMS) depending on the mode; in test mode a classification is computed where as in training mode the tuple is processed by classifier instance (MC-NN-x).

As can be seen in figure 15, ‘FilterTrainTest’ bolt performs an “all grouping” stream operation (replication) because the bolt classifiers (light blue) will use the same tuple. Depending on the mode different behavior is expected.

If training mode is enabled, one Classifier instance (light blue) will independently classify the tuple using the Micro-Cluster Nearest Neighbour algorithm [1]. Nevertheless, if test mode is enabled, each classifier computes a predicted label and another join is needed (grey - JOINCLASSIFIERS) which merges all the predicted labels from each classifier.

Finally, each joined tuple is passed into a last “ComputeMajorVote” bolt (light blue) that computes the major vote of the predicted labels and pushes the result into a Kafka output topic (green).

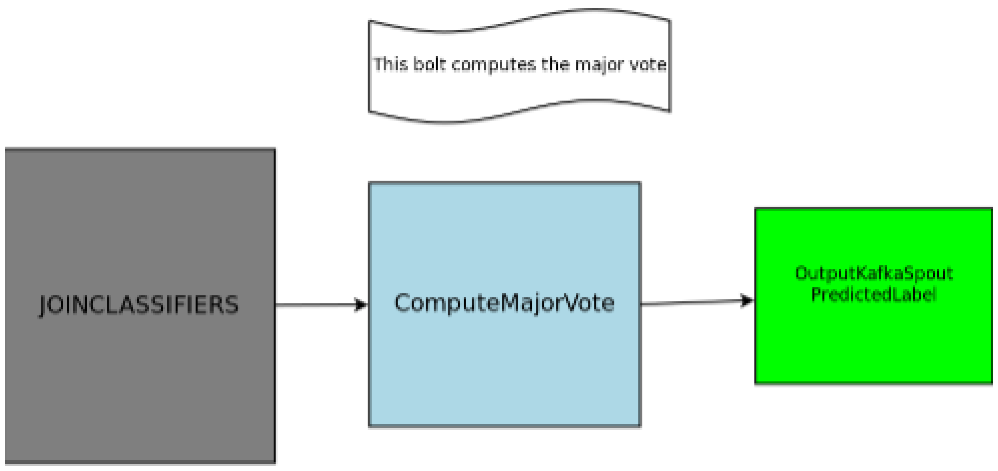


Figure 16: In test mode, a major vote is computed to assign a predited label to the new data instance.

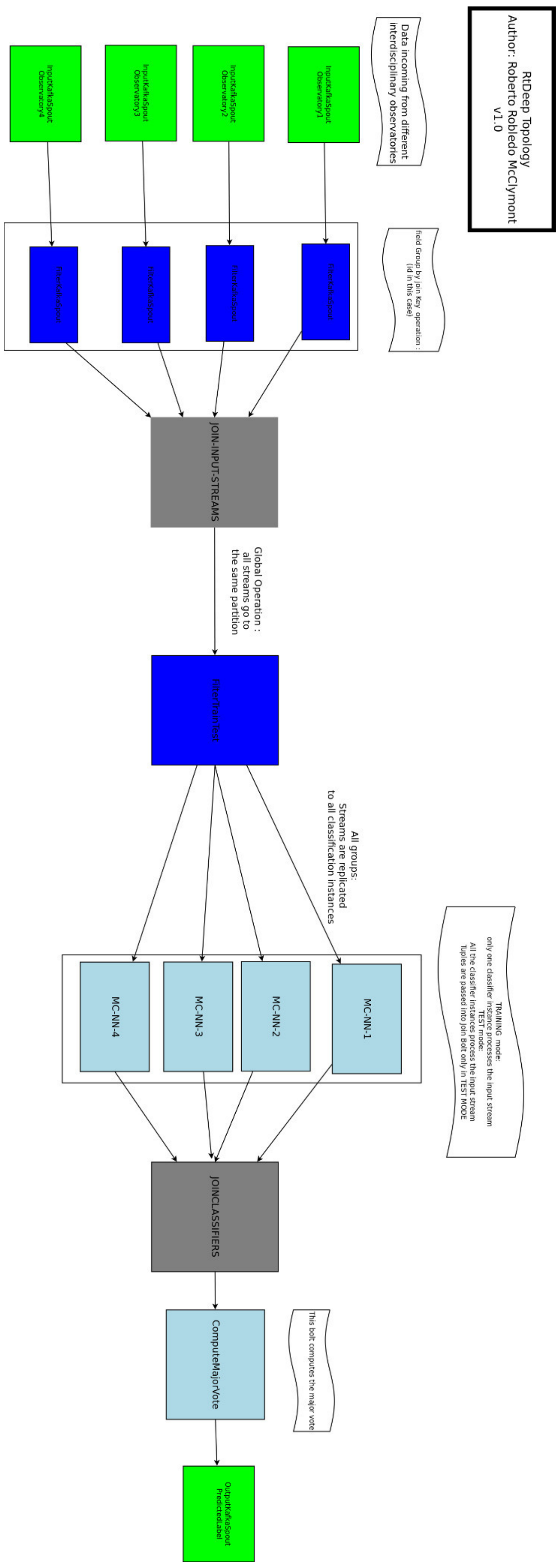

RtDeep Topology
Author: Roberto Robledo McClymont
v1.0
Data incoming from different interdisciplinary observatories
InputKafkaSpout Observatory1
InputKafkaSpout Observatory2
InputKafkaSpout Observatory3
InputKafkaSpout Observatory4
field Group by join Key operation : (id in this case)
FilterKafkaSpout
FilterKafkaSpout
FilterKafkaSpout
FilterKafkaSpout
JOIN-INPUT-STREAMS
Global Operation : all streams go to the same partition
filterTrainTest
All groups: Streams are replicated to all classification instances
TRAINING mode: only one classifier instance processes the input stream
TEST mode: All the classifier instances process the input stream
Tuples are passed into Join Bolt only in TEST MODE
MC-NN-1
MC-NN-2
MC-NN-3
MC-NN-4
JOINCLASSIFIERS
ComputeMajorVote
This bolt computes the major vote
OutputKafkaSpout PredictedLabel

Figure 17: RtDeep Storm topology

4. Deep Learning

Deep learning is part of machine learning, and it is a method based on learning data representation. Learning can be supervised, semi-supervised or unsupervised. RtDeep prototype uses supervised learning which analyzes the training data and produces an inferred function. Afterwards, this function can be used for mapping new examples.

It is important to highlight that this inferred function is based on the MC-NN (Micro Cluster Nearest Neighbour) which is the KNN algorithm adapted for a cluster environment [1] based on a feasibility study of a real-time KNN classifier [2]. Note that the MC-NN uses a technique originally developed for the data streaming cluster using Micro-Clusters [3].

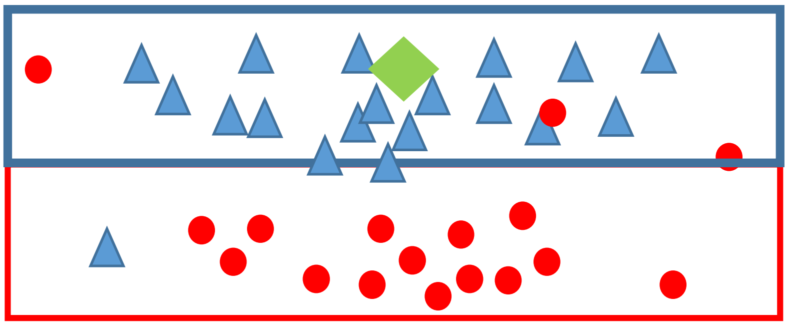


Figure 18: KNN algorithm

4.1 Micro-Cluster Nearest Neighbour

The MC-NN algorithm is a set of Micro-Clusters that computes its nearest neighbours for classification taking into account two variables; value and time stamp. The Micro-Cluster is composed of the following structure:

$$\langle CF2\ x, CF1\ x, CF1\ t, n, CL, \epsilon, \Theta, \alpha, \Omega \rangle$$

As can be seen in the notation above, CF stands for Cluster Feature, the superscripts x and t denote if the statistics are about values or time stamps respectively. Number 1 and 2 are used to denote if CF stores the sums of the feature values or the squared sum respectively. The number of data instances are stored in a scalar n. CL denotes class label, ϵ as error count, Θ as error threshold for splitting, α as initial time and Ω as a threshold for the Micro-Cluster's performance.

On the one hand, the value of each data instance is taken into account and a centroid is calculated for each Micro-Cluster by CF1/n and it is used afterwards to classify new data instance from the stream computing Euclidean distances between the new data instance and each Micro-Cluster centroid with the same class label. In addition, ϵ (error count) is initially set to '0' and incremented by 1 if the nearest MC (Micro Cluster) is a different class label or decremented if it is the same class label. Every time a new data instance is added in the training mode the following check is performed for each MC:

$$\epsilon > \Theta$$

If over time any error count of any MC reaches the error threshold, automatically the MC is split. Before making the division an evaluation noted in (Eq1) is computed for each particular attribute (x denotes a particular attribute from any of the four observatories):

$$\boldsymbol{Variance}[\boldsymbol{x}] = \sqrt{\frac{\boldsymbol{CF2^x}}{\boldsymbol{n}} - \left(\frac{\boldsymbol{CF1}}{\boldsymbol{n}}\right)^2}$$

(Eq1)

The splitting of the MC generates two MCs centred in the point of the parent MC attribute of the greatest variance while the parent MC is removed and the new MCs inherit all configuration values from the parent. The split attribute (with the largest variance), is altered by the variance value identified in the positive direction in one of the new Micro-Clusters and negatively in the other to ensure that future training will further re-position the two new Micro-Clusters better than the parent could do alone.

On the other hand, the time stamp is also taken into account using a Triangular Number with the following equation Eq2:

$$T = \left(\frac{(T^2 + T)}{2}\right)$$

(Eq2)

The use of Triangular Numbers gives more importance to recent instances over earlier ones added to the Micro-Cluster, as the time stamp value (T) is always increasing and MC-NN uses the sum of these incremental values. Triangular numbers assume that all Micro-Clusters were created at time stamp 1. To counter this, each Micro-Cluster keeps track of the time stamp when it was initialized (α). Following the same philosophy of evaluating the variance of attributes when a new instance has been added, another check is performed:

$$\text{Percentage MC} > \Omega$$

If the MC percentage drops below the performance threshold (Ω) the Micro-Cluster is deleted.

4.2 MC-NN parallel implementation

The underlying idea behind parallel implementation is that each node in a computer cluster is training MC individually and each computer node is computing its similarity to newly arrived labelled data instances. This parallel MC-NN implementation can be described in three steps: MC initialization, the training of MC and the prediction of the new arrival unlabeled data instances (testing).

The first step is related to MC initialization which defines values to the predefined values for error count (ϵ), error threshold (θ), the time stamp threshold (Ω), data stream parameters such as attribute numbers, class labels.

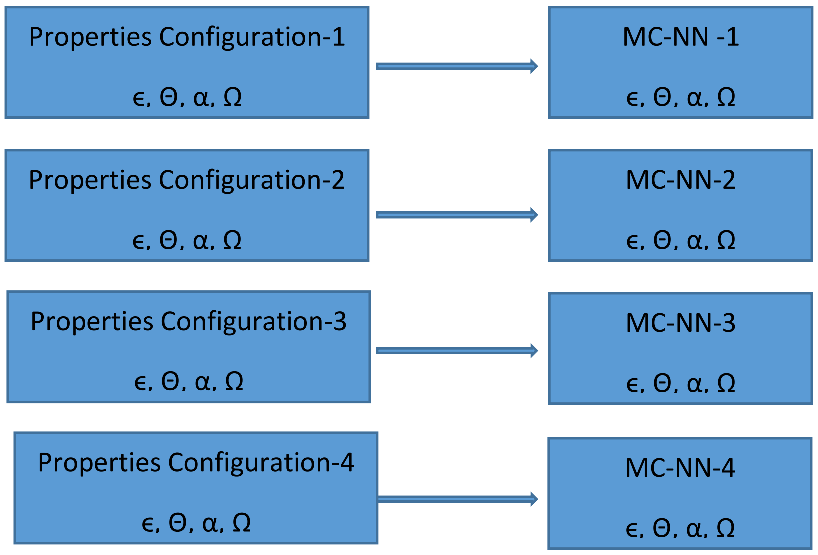


Figure 19: Initialisation setup of each MC-NN

The second step is the training of the individual nodes once a labelled training instance arrives the closest MC absorbs the new data instance described above and subsequently may split into more MC or delete older less participating clusters in order to adapt to concept drifts (values and time stamps).

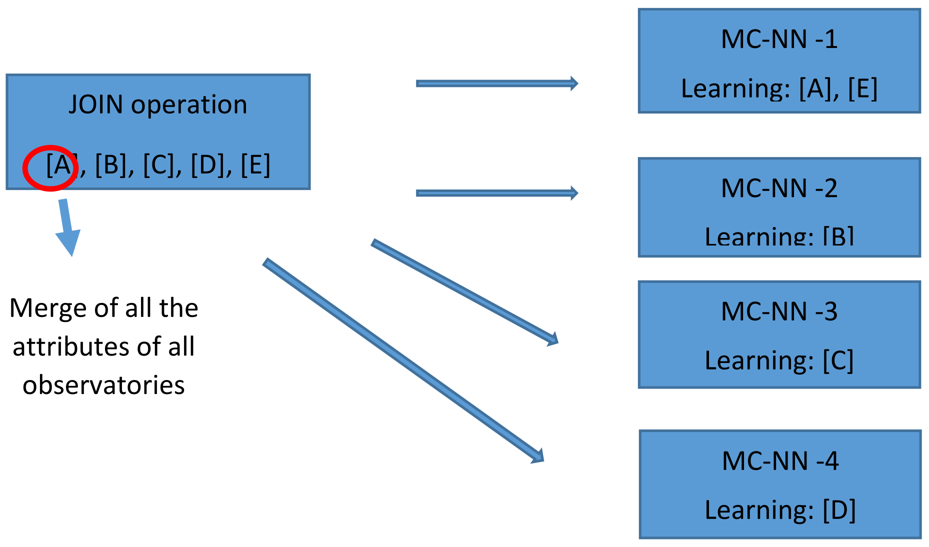


Figure 20: Training Distribution

The third step is the prediction of the unlabeled data instance using its local configuration of the Micro Cluster using the algorithm described above.

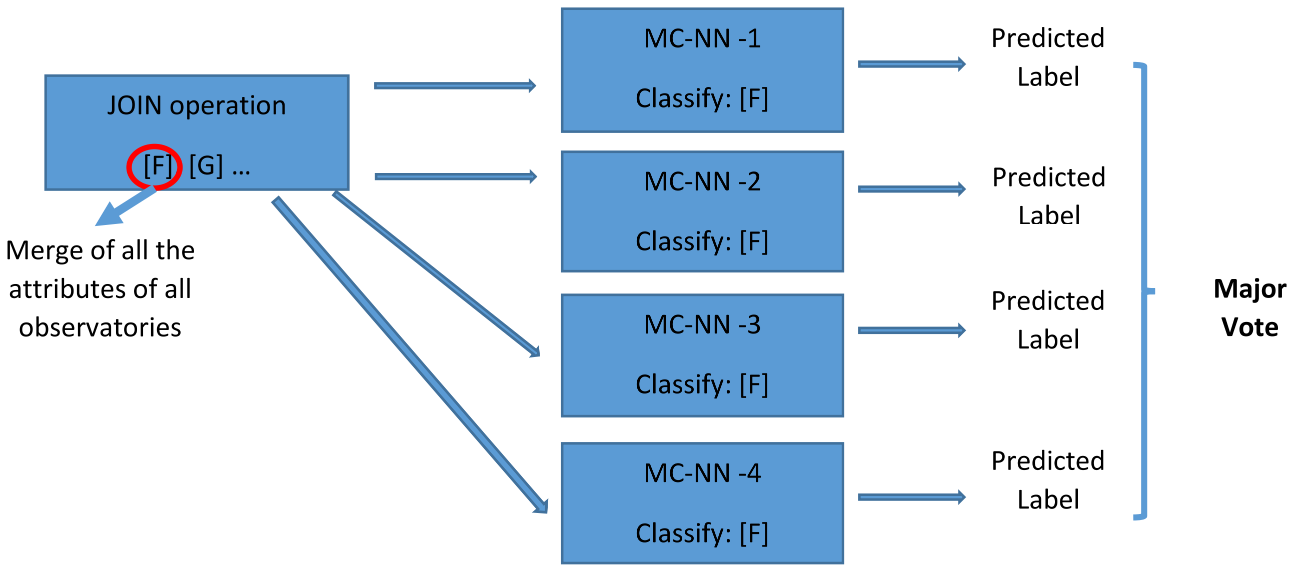


Figure 21: Test Cycle

5. Design

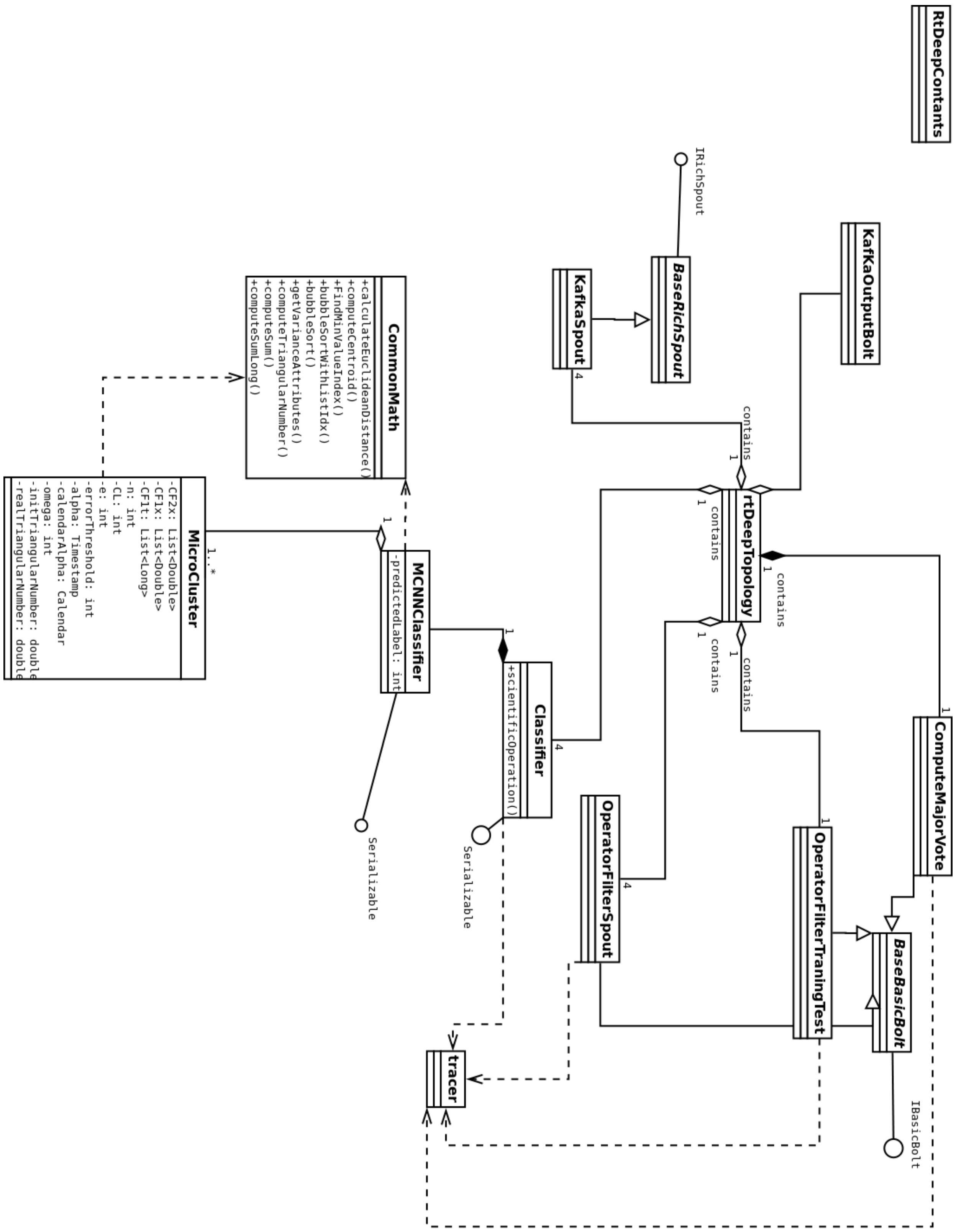


Figure 22: rtDeep Diagram Class

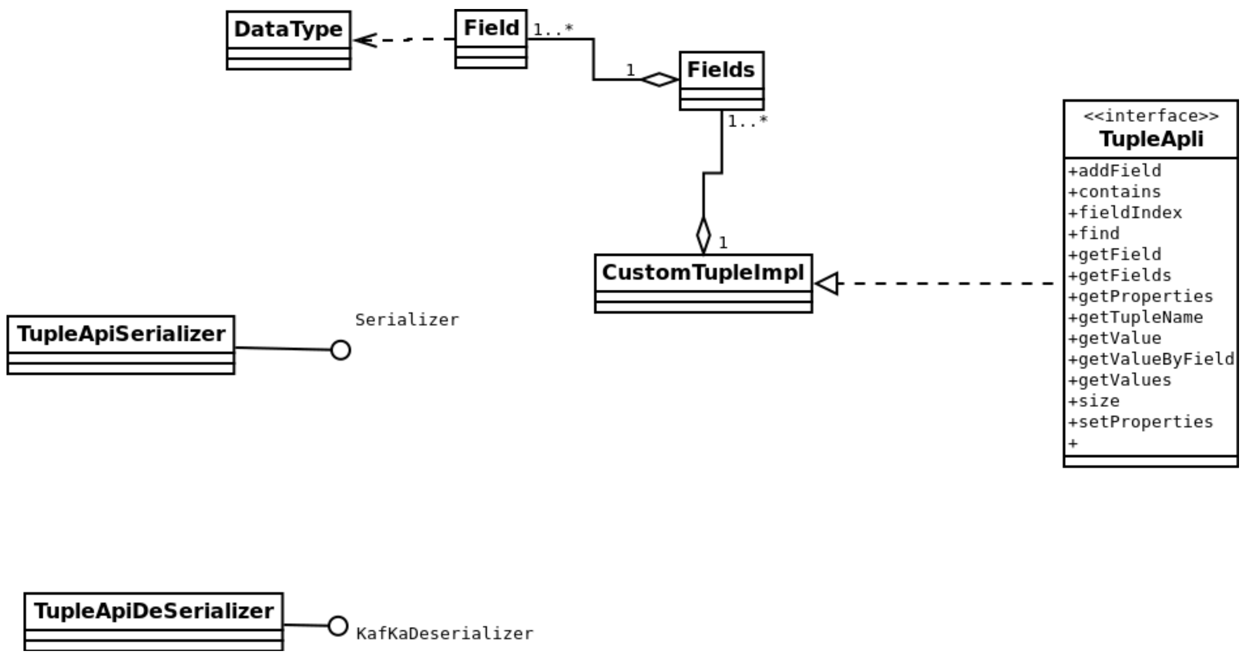


Figure 23: Tuple Diagram class (how data is passed within rtDeep prototype)

6. Implementation & Validation

As is noted in Section 1, the programming language used for implementing the rtDeep prototype is Java using the OOP paradigm. Nonetheless, bash and python scripting are used to check the synthetic input and output data.

Object-oriented programming (OOP) is a programming paradigm that consists of a procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which consists of interacting objects. The concept of "objects" which may contain data, in the form of fields known as attributes and a functional code in the form of procedures known as methods.

Validation is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It has been decided to split the validation into two main parts.

The first validation carried out is related to the topology implementation which focuses on validating that data are passed in the topology as expected. To do so, a trace analysis is done. This trace analysis consists of printing through the standard output each Storm component when a tuple is received or emitted. Finally, a parse is done in each Storm component in order to check if the inputs and outputs are correctly passed.

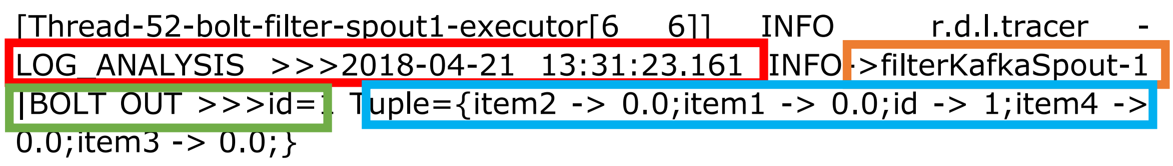
Taking into account that this first validation is done with 5 tuples (4 training / test) because it lets us check visually if data is passed in the expected way. The visual check can be split into several visual steps but for future work it could be interesting to automatize this procedure using an external tool that counts the number of inputs and outputs of each storm component. Two steps are done:

1) Copying the standard output into an external file.
2) Checking how many tuples have been received for each component (a tag LOG ANALYSIS with its date has been added to each print)

If the topology is run the following messages will appear:

```
 [Thread-52-bolt-filter-spout1-executor[6    6]]    INFO        r.d.l.tracer    -
LOG_ANALYSIS  >>>2018-04-21  13:31:23.161  INFO->filterKafkaSpout-1
|BOLT  IN  >>>id=1  Tuple={item2  ->  0.0;item1  ->  0.0;id  ->  1;item4  ->
0.0;item3 -> 0.0;}
```

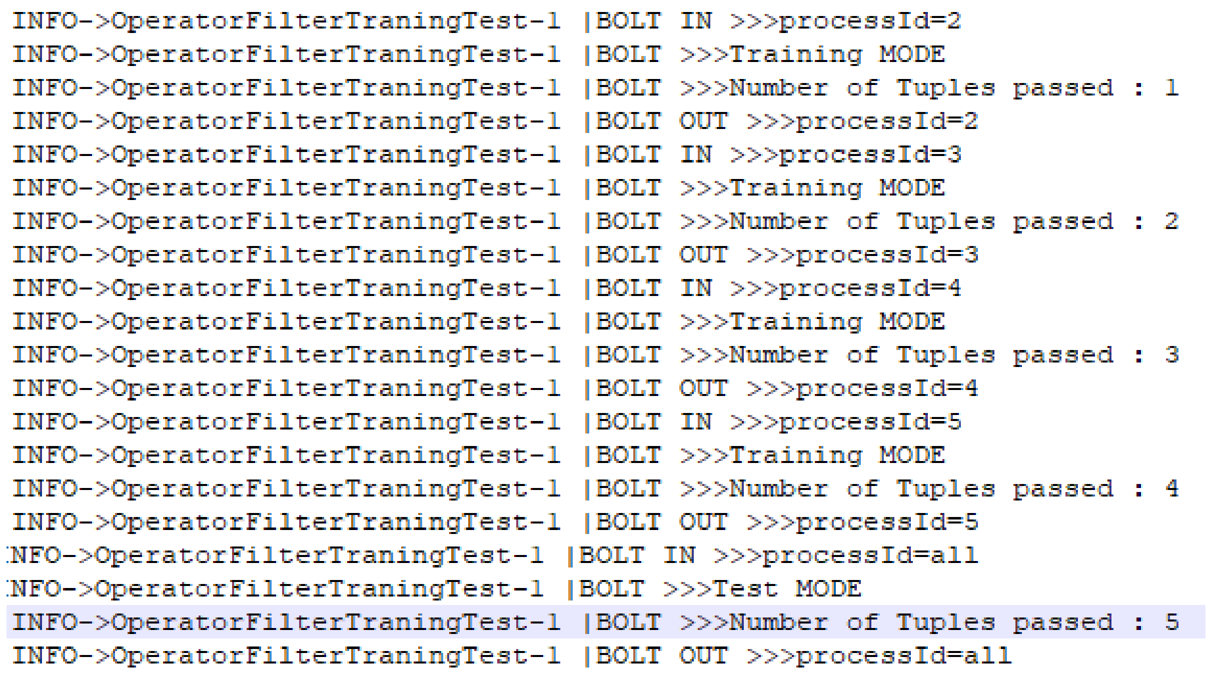
🟥 LOG ANALYSIS TAG with its date	🟩 Input tuple with its id	🟦 Id Storm Component	🟧 Content of the tuple



```
[Thread-52-bolt-filter-spout1-executor[6    6]]    INFO        r.d.l.tracer    -
LOG_ANALYSIS  >>>2018-04-21  13:31:23.161  INFO->filterKafkaSpout-1
|BOLT OUT >>>id=1. Tuple={item2 -> 0.0;item1 -> 0.0;id -> 1;item4 ->
0.0;item3 -> 0.0;}
```

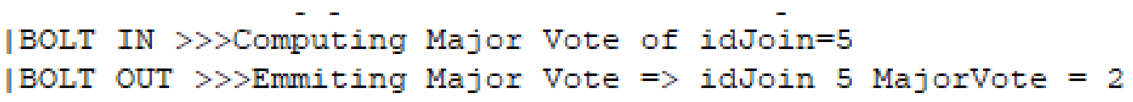
LOG ANALYSIS TAG with its date	Output tuple with its id	Id Storm Component	Content of the tuple

Moreover, some traces have been added in the OperatorFilterTrainingTest bolt to check the number of tuples passed and in which training or testing is.



```
INFO->OperatorFilterTraningTest-1 |BOLT IN >>>processId=2
INFO->OperatorFilterTraningTest-1 |BOLT >>>Training MODE
INFO->OperatorFilterTraningTest-1 |BOLT >>>Number of Tuples passed : 1
INFO->OperatorFilterTraningTest-1 |BOLT OUT >>>processId=2
INFO->OperatorFilterTraningTest-1 |BOLT IN >>>processId=3
INFO->OperatorFilterTraningTest-1 |BOLT >>>Training MODE
INFO->OperatorFilterTraningTest-1 |BOLT >>>Number of Tuples passed : 2
INFO->OperatorFilterTraningTest-1 |BOLT OUT >>>processId=3
INFO->OperatorFilterTraningTest-1 |BOLT IN >>>processId=4
INFO->OperatorFilterTraningTest-1 |BOLT >>>Training MODE
INFO->OperatorFilterTraningTest-1 |BOLT >>>Number of Tuples passed : 3
INFO->OperatorFilterTraningTest-1 |BOLT OUT >>>processId=4
INFO->OperatorFilterTraningTest-1 |BOLT IN >>>processId=5
INFO->OperatorFilterTraningTest-1 |BOLT >>>Training MODE
INFO->OperatorFilterTraningTest-1 |BOLT >>>Number of Tuples passed : 4
INFO->OperatorFilterTraningTest-1 |BOLT OUT >>>processId=5
NFO->OperatorFilterTraningTest-1 |BOLT IN >>>processId=all
NFO->OperatorFilterTraningTest-1 |BOLT >>>Test MODE
INFO->OperatorFilterTraningTest-1 |BOLT >>>Number of Tuples passed : 5
INFO->OperatorFilterTraningTest-1 |BOLT OUT >>>processId=all
```

Figure 24: Example traces added within operatorFilterTrainingTest



```
|BOLT IN >>>Computing Major Vote of idJoin=5
|BOLT OUT >>>Emmiting Major Vote => idJoin 5 MajorVote = 2
```

Figure 25: A major vote is computed taking into account all MC-NN instances

As can be seen in figures 24 and 25, 5 tuples are passed and successfully joined using their dentifier and finally one tuple (id=5) with its predicted label is emitted to the end of the topology. Lastly, the Kafka output topic must be checked if it is successfully filled in and to do so, a script has been implemented and is located in src/main/setUpKafka/consumeMsgOutput.sh

```
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-console-consumer.sh --topic result-label-predict
--from-beginning --zookeeper localhost:2181
```

If this script is run one message will be received containing predictedLabel=2

The second validation consists of validating to the MC-NN (Micro-Cluster Nearest Neighbor) implementation and this is done in several steps. In the first step, we must ensure that input synthetic data is properly generated using a multidimensional scaling (MDS) algorithm as can be seen in figure 12.

MDS visualises the level of similarity of in the individual cases of dataset. It refers to a set of related ordination techniques using information visualization, in particular to display the information contained in a distance matrix. The MDS algorithm aims to place each object in an N-dimensional space so that the between-object distances are preserved as well as possible. Therefore, a matrix will be created using the different incoming data from the different observatories with a size of [nobservations,nsamples] where *nobservations* refers to a specific frequency in time and *nsamples* refers to the joined list of attributes of all the observatories.

Once the input synthetic data is validated, output data must also be validated. Note that each classifier instance will compute a predicted label taking into account its own knowledge (each of them have trained with several tuples previously as is explained above in Section 4) and a final bolt will compute the major vote predicted label to finally assign it to the new data instance. Again, a trace analysis will be carried out using the printed information from the standard output. The objective of this trace is to retrieve the final predicted label of each test tuple in order to check if the predicted label matches its "real" label that is known in the generation of the input synthetic data.

7. Results

Validation techniques in machine learning are used to get the error rate of the learning model. If the volume of data is not large enough to be representative of the population, some validation techniques can be used such us cross validation which gives a more accurate estimate of the model prediction. Nonetheless, a simple validation technique is used.

Simple validation is a technique that is composed of two datasets: training dataset which also called "known data" is used for the learning model (MC-NN algorithm) and the second is the validation or testing set also called "unknown data" (or first seen data) which is used to evaluate the accuracy of the learning model.

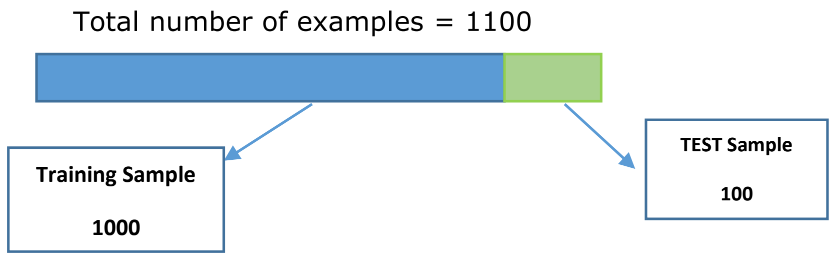
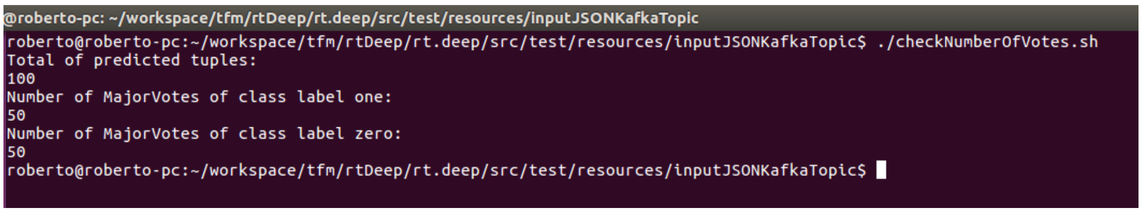


Figure 26: Experiment done with synthetic input data used in rtDeep prototype.

As can be seen in figure 26, a set of 1000 samples has been used by the MC-NN classifiers with the two possible scenarios as explained in section 2. As already mentioned the 'FilterTrainTest' bolt is responsible for distributing, in a balanced way, the new data instances into the classifiers which gives a total of 250 samples for each classifier. Then, a set of 100 samples are used to test the accuracy of the rtDeep prototype.

The test set has been sorted in order to simplify the process validation and this means that the first fifty samples belong to the first class (normal case) whereas the remaining belong to the second class (tsunami case).

In addition, a script which is called "checkNumberOfVotes.sh" has been implemented in order to count how many samples are classified as normal and the tsunami case and is located in the following path /src/test/resources/inputJSONKafkaTopic/



```
@roberto-pc: ~/workspace/tfm/rtDeep/rt.deep/src/test/resources/inputJSONKafkaTopic
roberto@roberto-pc:~/workspace/tfm/rtDeep/rt.deep/src/test/resources/inputJSONKafkaTopic$ ./checkNumberOfVotes.sh
Total of predicted tuples:
100
Number of MajorVotes of class label one:
50
Number of MajorVotes of class label zero:
50
roberto@roberto-pc:~/workspace/tfm/rtDeep/rt.deep/src/test/resources/inputJSONKafkaTopic$ 
```

Figure 27: Number of Votes for both scenarios (normal and tsunami)

As you can see in the image above, 50 samples are found for each class using 100 test instances.

1000 tuples were used to train the classifier instances and, in addition, 100 tuples were used to test the classifiers. For a faster validation, test data instances were sorted in the following order: the first 50 tuples belong to class label "0" and the remaining 50 belong to class label "1".

Due to the fact that test instances are sorted, a visual check in the standard output can be carried out; 1000 -1050 tuples should be classified as class label "0" and the remaining (1050-1100) should be classified as class label "1".

As can be seen in the following table, rtDeep prototype gives us 100% accuracy.

Class	**Accuracy** **Predicted/Real*100**
Zero (Normal Case)	(50/50*100)=100%
One (Tsunami Case)	(50/50*100)100%

Table 2: MC-NN accuracy using a set of 1000 Train samples and set of 100 Test samples.

8. Conclusions

A real-time prototype has been designed and implemented using the Apache Storm and Kafka framework and which is capable of classifying real-time data using the Micro Cluster Nearest Neighbour algorithm giving good results in terms of accuracy.

The design and its implementation has been done using the modular programming technique that emphasizes separating the functionality of a program into independent interchangeable modules as can be seen in the presented diagram class. Moreover, this technique will let us, in the case of future work improve the current deep learning technique or even to test further deep learning techniques.

As for future work, there are several things to tackle within the rtDeep topology scope:

First, the creation of an automatic tool that parses the rtDeep prototype log traces in order to rapidly parse the input/output tuples of the launched topology which could help us to increase the complexity of the topology with almost no effort.

Secondly, the refinement of the synthetic input data can be improved in two ways; adding more classes and/or including different statistical distributions which can help us to demonstrate the robustness of the learning model chosen (MC-NN).

Thirdly, the creation of an automatic tool that generates Storm components (Spouts, Bolts or even new Operators that extend from these) but reduces the time consumed when adding new components with their stream grouping.

Fourthly, the use of real data from different observatories instead of simulating input data with statistical distribution will make the prototype more attractive and it will move it from being a prototype to a "real" product.

Last but not least, the rtDeep prototype must be deployed in a cluster environment in order to take advantage of the horizontal scalability that Kafka and Storm provide.

As in all research studies, iterative processing must be carried out in order to enhance the deep algorithm and the topology presented during this final master project.

9. References

[1] Mark Tennant, Frederic Stahl, Omer Rana, João Bártolo Gomes, Scalable real-time classification of data streams with concept drift, Future Generation Computer Systems, Volume 75,2017, Pages 187-199,ISSN 0167-739X, https://doi.org/10.1016/j.future.2017.03.026.

[2] M. Tennant, F. Stahl, G. Di Fatta, J.B. Gomes, Towards a parallel computationally efficient approach to scaling up data stream classification, in: M. Bramer, M. Petridis (Eds.), Research and Development in Intelligent Systems XXXI, Springer International Publishing, 2014, pp. 51–65. http://dx.doi.org/10.1007/978-3-319-12069-0_4

[3] C. Aggarwal, J. Han, J. Wang, P. Yu, A framework for clustering evolving data streams, in: Proceedings of the 29th VLDB Conference, Berlin Germany, 2003

10. Annex

```
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class is in charge of classifying the input data using the following paper:
 *  Scalable real-time classification of data streams with concept drift
 */
package rt.deep.classifier;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;
import rt.deep.log.tracer;
import rt.deep.classifier.mcnn.MCNNClassifier;
import rt.deep.constants.RtDeepContants;
import rt.deep.tuple.Field;
import rt.deep.tuple.TupleApi;

public class Classifier extends BaseBasicBolt {

    private String idName="classifier";
    /** serial ID */
    private static final long serialVersionUID = -3726297716111144181L;
    /** logger*/
    private tracer logger = new tracer();
    //properties file
    private Properties prop= new Properties();
    // path properties file
    private String pathPropFile;
    //list of fields read from properties file
    private List<String> listInFieldsStr;
    //list of fields read from properties file
    private List<String> listOutFieldsStr;
    // compute flag
    boolean computeFlag;
    // Test mode: predicted label
    private int predictedLabel;
    // input tuples after join operation
    List<TupleApi> inputTuples = new ArrayList<TupleApi>();
    // identifer
    int idBolt;

    private String processId;

    // classifier MC-NN object that follows
    // * Scalable real-time classification of data stream with concept drift
    // * Mark Tennat, Frederic Stahl, Omer Rana, Joao Bartolo Gomes
    MCNNClassifier mcNN;
    //constructor
    public Classifier(String pathPropertiesFile,List<Integer>listClass,int id){
        this.pathPropFile = pathPropertiesFile;
        this.readPropertiesFile();
        // custom idName
        this.idName = this.idName + "-"+String.valueOf(id);
        this.idBolt=id;
        this.mcNN = new MCNNClassifier(listClass);
    }
    // add begin trace : BOLT
    private void logIn(String msg){
        this.logger.logInfo(this.idName +" |BOLT IN >>>"+msg);
    }
    // add begin trace : BOLT
    private void logOut(String msg){
        this.logger.logInfo(this.idName + " |BOLT OUT >>>" +msg);
    }
    // add begin trace : SPOUT
    private void logInfo(String msg){
        this.logger.logInfo(this.idName + " |BOLT >>>" +msg);
    }
    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {

        // get processId
        this.processId = (String)tuple.getValueByField("processId");
        TupleApi unWrappTuple =(TupleApi)tuple.getValueByField("tuple");
        String idJoin =(String)unWrappTuple.getValueByField(RtDeepContants.ID_JOIN_FIELD);
        if(!this.processId.equals(RtDeepContants.process_ID_ALL_BOLTS)){
            // check if idBolt corresponds with the remainder of processId
            if((this.idBolt%RtDeepContants.NUM_CLASSIFIER_INSTANCES) ==
            (Integer.valueOf(this.processId)%RtDeepContants.NUM_CLASSIFIER_INSTANCES)){
                this.computeFlag = true;
            }
            // this bolt will not proceed with this tuple
            else{
                this.computeFlag = false;
            }
        }
        else{
            this.computeFlag = true;
        }
        if(this.computeFlag){
            // do
            this.logIn("Tuple with id=" + idJoin +" is  processed by this classifier.");
            List<TupleApi> listTuples
            =(List<TupleApi>)unWrappTuple.getValueByField(RtDeepContants.LIST_TUPLE_FIELD);
            // follow MC-NN computation noted in the paper
            this.ScientificOperation(listTuples);
        }
        else{
            this.logIn("Tuple with id=" + idJoin +" is  not processed by this classifier.");
        }
        // only emit tuples in ALL MODE (TEST MODE)
        if(this.processId.equals(RtDeepContants.process_ID_ALL_BOLTS)){
            this.logOut("Emmiting prediction => idJoin " + idJoin + " predictedLabel = "+
            this.predictedLabel);
            collector.emit(new Values(idJoin,this.predictedLabel));
        }
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields(this.listOutFieldsStr));
    }
    @Override
    public void prepare(Map stormConf, TopologyContext context) {

    }
    private void readPropertiesFile(){
        this.logInfo("Reading properties file...");
        InputStream input = null;
        try {
            input = new FileInputStream(this.pathPropFile);
            // load a properties file
            this.prop.load(input);
        } catch (FileNotFoundException e) {
            this.logger.logError(e.toString());

        } catch (IOException e) {
            this.logger.logError(e.toString());
        }
        // retrieve configuration parameters
        this.listInFieldsStr =
        Arrays.asList(this.prop.getProperty("bolt.input.fields").split(","));
        this.listOutFieldsStr =
        Arrays.asList(this.prop.getProperty("bolt.output.fields").split(","));
        this.logInfo("Read done successfully");
    }
    private void ScientificOperation(List<TupleApi>listTuple){
        // convert tuples into list of doubles
        // get the new class label
        List<Double> newValues = this.encode(listTuple);
        this.logInfo("input values : " + newValues.toString());

        //new arrived unlabelled data
        if(this.processId.equals(RtDeepContants.process_ID_ALL_BOLTS)){
            // add new data instance to the Micro-Cluster
            this.predictedLabel = this.mcNN.computeNearestMCTestMode(newValues);
        }
        // training  mode
        else{
            int newCL = this.getClassLabelFromInputTuple(listTuple);
            this.logInfo("newTrainigCL = "+newCL);
            this.mcNN.computeNearestMCTrainingMode(newValues, newCL);
        }

    }
    // get class label from input tuple
    private int getClassLabelFromInputTuple(List<TupleApi>listTuple){
        String intStr;
        int newCL = -1;
        if(listTuple.size()>0){
            intStr = (String)listTuple.get(0).getValueByField(RtDeepContants.classLabel);
            newCL = Integer.valueOf(intStr);
        }
        return newCL;
    }
    // encode to what Classifier MC-NN is expecting (list of double each attribute
    corresponds an item from the input spouts
    private List<Double> encode(List<TupleApi>listTuple){
        List<Double> listValues = new ArrayList<Double>();
        for(int i = 0 ;i<listTuple.size();i++){
            // get Ituple
            TupleApi tuple = listTuple.get(i);
            List<Field> listField= tuple.getFields().toList();

            for(int idxField = 0; idxField<listField.size();idxField++){
                // get fields
                // get values from fields different from id and classlabel
                if(!listField.get(idxField).getName().equals(RtDeepContants.idTuple)
                        &&
                        !listField.get(idxField).getName().equals(RtDeepContants.classLabel)
                        ){
                    // get value from JSON
                    String val =
                    (String)tuple.getValueByField(listField.get(idxField).getName());
                    // cast
                    listValues.add(Double.valueOf(val));
                }
            }
        }
        return listValues;
    }
```

```
}
```

```java
package rt.deep.classifier.mcnn;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import rt.deep.common.math.CommonMath;

/**
 * This class implements the Micro-Cluster Nearest Neighbor (MC-NN) data stream
 * classifier from the following paper :
 * Scalable real-time classification of data stream with concept drift
 * Mark Tennat, Frederic Stahl, Omer Rana, Joao Bartolo Gomes
 * Author: Roberto Robledo McClymont
 * Version: 1.0
 *
 */
public class MCNNClassifier implements Serializable{
    // serial UID
    private static final long serialVersionUID = -6120406671124459388L;
    // list of MC
    List<MicroCluster> listMC = new ArrayList<MicroCluster>();
    public MCNNClassifier(List<Integer> listCL){
        for(int i = 0; i<listCL.size();i++){
            // init MC-NN with a list of MC with their class label
            this.listMC.add(new MicroCluster(listCL.get(i)));
        }
    }
    // TESTING MODE : Compute Euclidean Distance
    public int computeNearestMCTestMode(List<Double>newData){
        List<Double> distanceMC = new ArrayList<Double>();
        for(int i = 0; i<this.listMC.size();i++){
            // compute distance to each centroid
            List<Double> centroidCF1x = new ArrayList<Double>();
            // compute centroid CF1x
            centroidCF1x = CommonMath.computeCentroid(this.listMC.get(i).getCF1x(),
            this.listMC.get(i).getN());
            // compute Euclidean distance
            distanceMC.add(this.computeDistanceMCandInputData(newData,centroidCF1x));
        }
        int [] idxSortDist = CommonMath.bubbleSort(distanceMC);
        int predictedLabelClass;
        if(idxSortDist.length>0){
            predictedLabelClass = this.listMC.get(idxSortDist[0]).getCL();
        }else{
            // error
            predictedLabelClass=-1;
        }
        return predictedLabelClass;

    }
    // TRAINING MODE : new data arrival
    public void computeNearestMCTrainingMode(List<Double>newData,int newTrainigCL){
        // list MicroClusters
        List<Double> distanceMC = new ArrayList<Double>();
        List<Integer> listIdxSameCL = new ArrayList<Integer>();
        for(int i = 0; i<this.listMC.size();i++){
            if(this.listMC.get(i).getCL() == newTrainigCL){
                if(this.listMC.get(i).getN()>0){
                    // compute distance to each centroid
                    List<Double> centroidCF1x = new ArrayList<Double>();
                    // compute centroid CF1x
                    centroidCF1x = CommonMath.computeCentroid(this.listMC.get(i).getCF1x(),
                    this.listMC.get(i).getN());
                    // compute Euclidean distance
                    distanceMC.add(this.computeDistanceMCandInputData(newData,centroidCF1x));
                    listIdxSameCL.add(i);
                }
                // fill in first case
                else{
                    this.listMC.get(i).addNewInstance(newData);
                    // do not continue within the loop
                    break;
                }

            }


        }
        //nominal case
        if(distanceMC.size()>0){
            Integer [] listIndexes =
            CommonMath.bubbleSortWithListIdx(distanceMC,listIdxSameCL);
            // update MicroCluster error count
            this.updateMCError(newData,newTrainigCL,listIndexes);
            // if over time MC error e > errorThreshold then the MC is split
            this.checkAndUpdateMCErrorThreshold();
            // compute each MC participation percentage
            // if either of these percentages drops below the performance thresholds(omega)
            the MC is deleted.
            this.computeMCParticipationPercentage();
        }

    }
    // compute Euclidean distance : it has been demonstrated that works faster than other
    measures
    public double computeDistanceMCandInputData(List<Double>newData,List<Double> listCF1x){
        return CommonMath.calculateEuclideanDistance(listCF1x,newData);
    }
    private void updateRemoveErrorCounter(int index){
        int errorCount= this.listMC.get(index).getE();
        errorCount = errorCount - 1;
        // update error count
        this.listMC.get(index).setE(errorCount);
    }
    private void updateAddErrorCounter(int index){
        int errorCount= this.listMC.get(index).getE();
        errorCount = errorCount + 1;
        // update error count
        this.listMC.get(index).setE(errorCount);
    }
    // update error count for each microCluster
    public void updateMCError(List<Double>newData, int newTrainigCL, Integer [] listIndexes){


        // sorted distances and indexes by bubble algorithm
        // two possible scenarios :
        //scenario1 : if the nearest MC is of the same label as the training instance
        //then the instance is incrementally added to the MC
        // scenario 2 : if the nearest M is of a different CL, then the training instance is
        incrementally
        // added to the nearest MC that matches the training instance's class label.
        //However, the error count e of both MC are incremented
        // e = e-1
        boolean nearest = true;
        for(int i = 0 ;i<listIndexes.length;i++){
            int mcCL = this.listMC.get(listIndexes[i]).getCL();
            // scenario 1
            if(nearest){
                //scenario1 : if the nearest MC is of the same label as the training
                instance then the instance is incrementally added to the MC
                // e = e-1
                if(mcCL==newTrainigCL){
                    this.updateRemoveErrorCounter(listIndexes[i]);
                    // add new instance data into the MC
                    // n is incremented by 1 and the sum of time stamps (CF1t)
                    // is incremented by the new time stamp arrival value(T)
                    // the Triangular Number deltaT Eq(2) from the paper of this time stamp
                    // will give an upper bound to the maximum possible value of CF1t
                    // Triangular numbers gives more importance to recent instances
                    // over earlier ones added to the Micro-Cluster
                    // assumption : all MCs were created at time stamp 1 (alpha)
                    this.listMC.get(listIndexes[i]).addNewInstance(newData);
                }
                else{
                    this.updateAddErrorCounter(listIndexes[i]);
                }
            }
            // scenario 2
            else{
                if(mcCL==newTrainigCL){
                    this.updateAddErrorCounter(listIndexes[i]);
                    // add new instance data into the MC
                    // n is incremented by 1 and the sum of time stamps (CF1t)
                    // is incremented by the new time stamp arrival value(T)
                    // the Triangular Number deltaT Eq(2) from the paper of this time stamp
                    // will give an upper bound to the maximum possible value of CF1t
                    // Triangular numbers gives more importance to recent instances
                    // over earlier ones added to the Micro-Cluster
                    // assumption : all MCs were created at time stamp 1 (alpha)
                    this.listMC.get(listIndexes[i]).addNewInstance(newData);
                    break; // found closest Micro-cluster with same class label
                }
            }
            nearest = false;
        }

    }
    // if over time a Micro-Cluster's error counter reaches a error threshold then the MC is
    split.
    // this is done by evaluating the MC dimensions for the size of its variance (attribute
    with greatest variance)
    // two MC are created and inherit all configuration values from the parent. Therefore,
    all future classifications made by
    // these new MC will be the same class label
    // The assumption behind this way of splitting attributes is that a larger variance
    value of one attribute over another
    // indicates tat a greater range of values have been seen for this attribute. Therefore,
    the attribute may contribute
    // towards miss-classifications.
    // Initially populated with the parent's internal/center data(CF1x)
    public void checkAndUpdateMCErrorThreshold(){
        for(int i = 0; i<this.listMC.size();i++){
            if(this.listMC.get(i).getE()>this.listMC.get(i).getErrorThreshold()){
                // compute variance for each attribute
                List<Double> listVarianceAtt = CommonMath.getVarianceAttributes
                        (this.listMC.get(i).getCF1x(),this.listMC.get(i).getCF2x());

                // split MC into two MC
                // The split attribute with the largest variance
                //is altered by the variance value in positive and negative direction
                // index in ascending order
                int [] indexSortVariance = CommonMath.bubbleSort(listVarianceAtt);
                // largest variance last index
                int largestVarianceIdx = indexSortVariance[indexSortVariance.length-1];
                MicroCluster copyMC = new MicroCluster(this.listMC.get(i));

                // add variance positive variance value into the actual MC
                List<Double> tmpCF1 = new ArrayList<Double>();
                for(int idxCF = 0; idxCF <this.listMC.get(i).getCF1x().size();idxCF++){
                    tmpCF1.add(this.listMC.get(i).getCF1x().get(idxCF) +
                    listVarianceAtt.get(largestVarianceIdx));
                }
                this.listMC.get(i).setCF1x(tmpCF1);
                List<Double> tmpCF2 = new ArrayList<Double>();
                for(int idxCF = 0; idxCF <this.listMC.get(i).getCF2x().size();idxCF++){
                    tmpCF2.add(this.listMC.get(i).getCF2x().get(idxCF) +
                    listVarianceAtt.get(largestVarianceIdx));
                }
                this.listMC.get(i).setCF2x(tmpCF2);
                tmpCF1.clear();
                tmpCF2.clear();

                // add variance negative variance value to the largest variance attribute
                for(int idxCF = 0; idxCF <copyMC.getCF1x().size();idxCF++){
                    tmpCF1.add(copyMC.getCF1x().get(idxCF) +
                    listVarianceAtt.get(largestVarianceIdx));
                }
                copyMC.setCF1x(tmpCF1);

                for(int idxCF = 0; idxCF <copyMC.getCF2x().size();idxCF++){
                    tmpCF2.add(copyMC.getCF2x().get(idxCF) +
                    listVarianceAtt.get(largestVarianceIdx));
                }
                copyMC.setCF2x(tmpCF2);
                tmpCF1.clear();
                tmpCF2.clear();
            }
        }
    }
    // The lower the value of CF1t is from the Triangular Number the poorer the MC has been
    participating in the stream classification
    // The use of Triangular Number give more importance to recent instances over earlier ones
    public void computeMCParticipationPercentage(){
        for(int i = 0; i<this.listMC.size();i++){
            List<Long> CF1t = new ArrayList<Long> ();
            CF1t = this.listMC.get(i).getCF1t();
            if(CF1t.size()>0){
                double lastTimeStampValue = CF1t.get(CF1t.size()-1);
                // compute each MC real Triangular Number

                this.listMC.get(i).computeRealTriangularNumber(CommonMath.computeTriangularNum
                ber(lastTimeStampValue));
                long sum = CommonMath.computeSumLong(CF1t);
                // sum * 100 / real triangular number [%]
                if(((sum*100.0)/this.listMC.get(i).getRealTriangularNumber())>
                this.listMC.get(i).omega){
                    // delete MC due to low participation percentage
                    this.listMC.remove(i);
                }
            }

        }
    }
}
```

```
package rt.deep.classifier.mcnn;

import java.io.Serializable;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Date;
/**
 * This class implements the Micro-Cluster Nearest Neighbour (MC-NN) data stream
 * classifier from the following paper :
 * Scalable real-time classification of data stream with concept drift
 * Mark Tennat, Frederic Stahl, Omer Rana, Joao Bartolo Gomes
 * Author: Roberto Robledo McClymont
 * Version: 1.0
 *
 */
public class MicroCluster implements Serializable {

    // serial UID
    private static final long serialVersionUID = -3133942430933789129L;
    // CF = Cluster Feature
    // the sum of the squares of the attributes are maintained in a vector CF2x
    List<Double> CF2x = new ArrayList<Double>();
    // the sum of the values in a vector CF1x
    // calculates the locality and boundary of the Micro-Clusters
    List<Double> CF1x = new ArrayList<Double>();
    // the sum of the time stamps in a vector :
    //determines the recently of the data summarized in the cluster
    List<Long> CF1t = new ArrayList<Long>();

    // number of data instances
    int n;

    // class label
    int CL;

    // error count
    int e;
    // error threshold for splitting : it is expected than a low errorThres will cause
    // the algorithm to adapt to changes faster, but will be more susceptible to noise
    int errorThreshold;
    // initial time stamp
    Timestamp alpha;
    // assign time to calendar
    Calendar calendarAlpha;
    // threshold for the MC performance
    int omega;

    // initial triangular number for each microCluster
    double initTriangularNumber;
    // real triangular number
    double realTriangularNumber;

    public double getInitTriangularNumber() {
        return this.initTriangularNumber;
    }
    public void setInitTriangularNumber(double initTriangularNumber) {
        this.initTriangularNumber = initTriangularNumber;
    }
    // default constructor
    public MicroCluster(){
        // initially set to 0
        this.e = 0;
        this.n = 0;
        this.initTriangularNumber = 1;
        this.realTriangularNumber = 0;
        this.errorThreshold = 10; // max error count for a cluster
        this.omega = 50; // %50 min participation percentage
        this.CL = 0; // default

        // create alpha initial timeStamp
        Date date = new java.util.Date();
        this.alpha = new Timestamp(date.getTime());

        // create a calendar and assign it the same time
        this.calendarAlpha = Calendar.getInstance();
        this.calendarAlpha.setTimeInMillis(this.alpha.getTime());
    }
    // copy constructor
    public MicroCluster(MicroCluster original){
        this.e = original.getE();
        this.n = original.getN();
        this.initTriangularNumber = original.getInitTriangularNumber();
        this.realTriangularNumber = original.getRealTriangularNumber();
        this.errorThreshold = original.getErrorThreshold(); // max error count for a cluster
        this.omega = original.getOmega(); // %50 min participation percentage
        this.CL = original.getCL(); // default

        this.alpha = original.getAlpha();
        // create a calendar and assign it the same time
        this.calendarAlpha = original.getCalendarAlpha();
        // the sum of the squares of the attributes are maintained in a vector CF2x
        this.CF2x = original.getCF2x();
        // the sum of the values in a vector CF1x
        // calculates the locality and boundary of the Micro-Clusters
        this.CF1x = original.getCF1x();
        // the sum of the time stamps in a vector :
        //determines the recently of the data summarized in the cluster
        this.CF1t = original.getCF1t();
    }
    // constructor defining which is the class of the Micro-Cluster
    public MicroCluster(int classLabel){
        // initially set to 0
        this.e = 0;
        this.n = 0;
        this.initTriangularNumber = 1;
        this.realTriangularNumber = 0;
        this.errorThreshold = 10; // max error count for a cluster
        this.omega = 50; // %50 min participation percentage
        this.CL = classLabel;

        // create alpha initial timeStamp
        Date date = new java.util.Date();
        this.alpha = new Timestamp(date.getTime());

        // create a calendar and assign it the same time
        this.calendarAlpha = Calendar.getInstance();
        this.calendarAlpha.setTimeInMillis(this.alpha.getTime());
    }
    public Calendar getCalendarAlpha() {
        return this.calendarAlpha;
    }
    public void setCalendarAlpha(Calendar calendarAlpha) {
        this.calendarAlpha = calendarAlpha;
    }
    // add new instance
    public void addNewInstance(List<Double>newData){
        // first case
        ArrayList<Double> myTmpList = new ArrayList<Double>();
        if(this.CF1x.size()==0){
            this.CF1x = newData;
        }else{
            for(int i = 0; i<newData.size(); i++){
                myTmpList.add(newData.get(i)+ this.CF1x.get(i));
            }
            this.CF1x = myTmpList;
        }
        if(this.CF2x.size()==0){
            for(int i = 0; i<newData.size(); i++){
                this.CF2x.add(Math.pow(newData.get(i),2.0));
            }
        }else{
            myTmpList.clear();
            for(int i = 0; i<newData.size(); i++){
                myTmpList.add(Math.pow(newData.get(i),2.0)+ Math.pow(this.CF2x.get(i),2.0));
            }
            this.CF2x = myTmpList;
        }

        // create a  second time stamp
        Timestamp newTimeStamp = new Timestamp(this.calendarAlpha.getTime().getTime());

        // get time difference from init timeStamp in milliseconds
        long milliseconds = newTimeStamp.getTime() - this.alpha.getTime();
        if(this.CF1t.size()==0){
            this.CF1t.add((long) 1);
            this.CF1t.add(milliseconds);
        }else{
            this.CF1t.add(milliseconds);
        }
        // update number of instance in a Micro-Cluster
        this.n= this.n +1 ;
    }
    public void computeRealTriangularNumber(double triangularValue){
        this.realTriangularNumber = triangularValue - this.initTriangularNumber;
    }
    public double getRealTriangularNumber() {
        return this.realTriangularNumber;
    }
    public void setRealTriangularNumber(double realTriangularNumber) {
        this.realTriangularNumber = realTriangularNumber;
    }
    public List<Double> getCF2x() {
        return this.CF2x;
    }
    public void setCF2x(List<Double> cF2x) {
        this.CF2x = cF2x;
    }
    public List<Double> getCF1x() {
        return this.CF1x;
    }
    public void setCF1x(List<Double> cF1x) {
        this.CF1x = cF1x;
    }
    public List<Long> getCF1t() {
        return this.CF1t;
    }
    public void setCF1t(List<Long> cF1t) {
        this.CF1t = cF1t;
    }
    public int getN() {
        return this.n;
    }
    public void setN(int n) {
        this.n = n;
    }
    public int getCL() {
        return this.CL;
    }
    public void setCL(int cL) {
        this.CL = cL;
    }
    public int getE() {
        return this.e;
    }
    public void setE(int e) {
        this.e = e;
    }
    public int getErrorThreshold() {
        return this.errorThreshold;
    }
    public void setErrorThreshold(int errorThreshold) {
        this.errorThreshold = errorThreshold;
    }
    public Timestamp getAlpha() {
        return this.alpha;
    }
    public void setAlpha(Timestamp alpha) {
        this.alpha = alpha;
    }
    public int getOmega() {
        return this.omega;
    }
    public void setOmega(int omega) {
        this.omega = omega;
    }


}
```

```java
package rt.deep.common.math;

import java.util.ArrayList;
import java.util.List;

public class CommonMath {
    public static double calculateEuclideanDistance(List<Double> array1, List<Double> array2)
    {
        double Sum = 0.0;
        for(int i=0;i<array1.size();i++) {
            Sum = Sum + Math.pow((array1.get(i)-array2.get(i)),2.0);
        }
        return Math.sqrt(Sum);
    }
    // compute centroid of an attribute
    public static List<Double> computeCentroid(List<Double> listValues,int n){
        List<Double> listCentroid = new ArrayList<Double>();
        for(int idx = 0; idx<listValues.size();idx++){
            listCentroid.add(listValues.get(idx)/n);
        }
        return listCentroid;
    }
    // find index min value
    public static int FindMinValueIndex (List<Double> arr1){//start method
        int index = 0;
        Double min = arr1.get(index);
        for (int i=1; i<arr1.size(); i++){
            if (arr1.get(i)< min ){
                min = arr1.get(i);
                index = i;
            }
        }
        return index ;
    }
    // sort by bubble algorithm using a list of index
    public static Integer [] bubbleSortWithListIdx(List<Double> distances,List<Integer>
    listIdxSameCL) {

        // convert list into array
        double[] numArray =  new double[distances.size()];
        Integer [] listIndex = new Integer[listIdxSameCL.size()];
        for(int i = 0; i<distances.size();i++){
            numArray[i] = distances.get(i);
            listIndex[i] = listIdxSameCL.get(i);
        }
        int n = numArray.length;
        double temp = 0;
        int tempIdx;
        for (int i = 0; i < n; i++) {
            for (int j = 1; j < (n - i); j++) {
                if (numArray[j - 1] > numArray[j]) {
                    temp = numArray[j - 1];
                    tempIdx = listIndex[j-1];
                    numArray[j - 1] = numArray[j];
                    listIndex[j-1] = listIndex[j];
                    numArray[j] = temp;
                    listIndex[j] = tempIdx;
                }
            }
        }
        return listIndex;
    }
    // sort distances and return a list of indexes (using bubble algorithm)
    public static int [] bubbleSort(List<Double> distances) {

        // convert list into array
        double[] numArray =  new double[distances.size()];
        for(int i = 0; i<distances.size();i++){
            numArray[i] = distances.get(i);
        }
        int n = numArray.length;
        double temp = 0;
        int tempIdx;
        int [] listIndex = new int[n];
        for(int i = 0; i<numArray.length;i++){
            listIndex[i] = i;
        }
        for (int i = 0; i < n; i++) {
            for (int j = 1; j < (n - i); j++) {
                if (numArray[j - 1] > numArray[j]) {
                    temp = numArray[j - 1];
                    tempIdx = listIndex[j-1];
                    numArray[j - 1] = numArray[j];
                    listIndex[j-1] = listIndex[j];
                    numArray[j] = temp;
                    listIndex[j] = tempIdx;
                }
            }
        }
        return listIndex;
    }
    // compute variance following Eq(1) noted in the paper
    public static List<Double> getVarianceAttributes(List<Double> CF1, List<Double> CF2){
        int n =  CF1.size();
        List<Double> variance = new ArrayList<Double>();
        for(int i = 0; i<CF1.size();i++){
            // compute variance for each attribute
            variance.add(Math.sqrt( (CF2.get(i)/n) - Math.pow((CF1.get(i)/n),2.0)));
        }
        return variance;
    }
    // compute a triangular number used to give more importance to the latest values
    public static double computeTriangularNumber(double timeStampValue){
        // Triangular Number deltaT = ((T^2+T)/2)
        return ((Math.pow(timeStampValue, 2)+timeStampValue)/2);
    }
    // compute the sum of a list of values
    public static double computeSum(List<Double> values){
        double sum=0.;
        for(int i = 0; i<values.size();i++){
            sum +=values.get(i);
        }
        return sum;
    }
    // compute the sum of a list of values
    public static Long computeSumLong(List<Long> values){
        Long sum=(long) 0;
        for(int i = 0; i<values.size();i++){
            sum +=values.get(i);
        }
        return sum;
    }
}
```

```java
package rt.deep.constants;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class is used to define constants in RtDeep prototype
 */
public final class RtDeepContants {
    public static int RT_DEEP_NUM_TRAINING_SET = 1000;
    public static String process_ID_ALL_BOLTS = "all";
    public static String ID_JOIN_FIELD = "idJoin";
    public static String LIST_TUPLE_FIELD = "listTuples";
    public static int NUM_CLASSIFIER_INSTANCES = 4;
    public static String idTuple = "id";
    public static String classLabel = "classlabel";
}
```

```
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class is in charge of filtering KafkaSpouts that are
 *  coming from different interdisciplinary observatories
 */
package rt.deep.filter;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Properties;

import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;
import rt.deep.log.tracer;
import rt.deep.tuple.TupleApi;

public class OperatorFilterSpout extends BaseBasicBolt {
    private String idName="filterKafkaSpout";
    /** serial ID */
    private static final long serialVersionUID = -3726297716111144181L;
    /** logger*/
    private tracer logger = new tracer();
    //properties file
    private Properties prop= new Properties();
    // path properties file
    private String pathPropFile;
    //list of fields read from properties file
    private List<String> listInFieldsStr;
    //list of fields read from properties file
    private List<String> listOutFieldsStr;
    //constructor
    public OperatorFilterSpout(String pathPropertiesFile,int id){
        this.pathPropFile = pathPropertiesFile;
        this.readPropertiesFile();
        // custom idName
        this.idName = this.idName + "-"+String.valueOf(id);
    }
    @Override
    public void prepare(Map stormConf, TopologyContext context) {

    }
    // add begin trace : BOLT
    private void logOut(String msg){
        this.logger.logInfo(this.idName +" |BOLT OUT >>>"+msg);
    }
    // add begin trace : BOLT
    private void logIn(String msg){
        this.logger.logInfo(this.idName +" |BOLT IN >>>"+msg);
    }
    // add begin trace : SPOUT
    private void logInfo(String msg){
        this.logger.logInfo(this.idName +" |BOLT >>>" +msg);
    }
    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {

        TupleApi varTuple = (TupleApi)tuple.getValue(0);
        // input tuple
        String id =(String) varTuple.getValueByField("id");
        this.logIn("id="+ id + " Tuple="+ varTuple.toString());
        this.logOut("id="+ id + " Tuple="+ varTuple.toString());
        collector.emit(new Values(id,varTuple));
    }
    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields(this.listOutFieldsStr));
    }
    private void readPropertiesFile(){
        this.logInfo("Reading properties file...");
        InputStream input = null;
        try {
            input = new FileInputStream(this.pathPropFile);
            // load a properties file
            this.prop.load(input);
        } catch (FileNotFoundException e) {
            this.logger.logError(e.toString());

        } catch (IOException e) {
            this.logger.logError(e.toString());
        }
        // retrieve configuration parameters
        this.listInFieldsStr =
        Arrays.asList(this.prop.getProperty("bolt.input.fields").split(","));
        this.listOutFieldsStr =
        Arrays.asList(this.prop.getProperty("bolt.output.fields").split(","));
        this.logInfo("Read done successfully");
    }

}
```

```
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class is in charge of Filtering data instances to be used
 *  in training mode or test mode
 */
package rt.deep.filter;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;
import rt.deep.log.tracer;
import rt.deep.utils.GeneralTools;
import rt.deep.constants.RtDeepContants;
import rt.deep.tuple.TupleApi;

public class OperatorFilterTraningTest extends BaseBasicBolt {
    //counter tuples change from training to test mode
    private int counterTuples;
    // counter bolt
    private Integer counterBolt;
    // identifier
    private String idName="OperatorFilterTraningTest";
    /** serial ID */
    private static final long serialVersionUID = -3726297716111144181L;
    /** logger*/
    private tracer logger = new tracer();
    //properties file
    private Properties prop= new Properties();
    // path properties file
    private String pathPropFile;
    //list of fields read from properties file
    private List<String> listInFieldsStr;
    //list of fields read from properties file
    private List<String> listOutFieldsStr;
    //constructor
    public OperatorFilterTraningTest(String pathPropertiesFile,int id){
        this.counterTuples = 0;
        this.counterBolt = 1;
        this.pathPropFile = pathPropertiesFile;
        this.readPropertiesFile();
        // custom idName
        this.idName = this.idName + "-"+String.valueOf(id);
    }
    @Override
    public void prepare(Map stormConf, TopologyContext context) {

    }
    // add begin trace : BOLT
    private void logOut(String msg){
        this.logger.logInfo(this.idName +" |BOLT OUT >>>"+msg);
    }
    // add begin trace : BOLT
    private void logIn(String msg){
        this.logger.logInfo(this.idName +" |BOLT IN >>>"+msg);
    }
```

```java
    // add begin trace : SPOUT
    private void logInfo(String msg){
        this.logger.logInfo(this.idName + " |BOLT >>>" +msg);
    }
    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {

        TupleApi wrappedTuple = GeneralTools.WrappTuples(tuple);
        String processId;
        // reset counter Bolt (training mode)
        // TRAINING MODE
        if(this.counterTuples>=RtDeepContants.RT_DEEP_NUM_TRAINING_SET){
            // all
            processId = RtDeepContants.process_ID_ALL_BOLTS;
        }
        // TEST MODE
        else{
            if(this.counterBolt>RtDeepContants.NUM_CLASSIFIER_INSTANCES){
                this.counterBolt = 1;
            }else{
                this.counterBolt = this.counterBolt + 1;
            }
            // select which bolt will use the data instance to train
            processId = String.valueOf(new Integer(this.counterBolt));
        }
        this.logIn("processId="+ processId);
        if(this.counterTuples<RtDeepContants.RT_DEEP_NUM_TRAINING_SET){
            this.logInfo("Training MODE");
        }
        else{
            this.logInfo("Test MODE");
        }

        this.counterTuples++;
        this.logInfo("Number of Tuples passed : " + this.counterTuples);
        this.logOut("processId="+ processId);
        // emit tuples to all the Classifier instances (all grouping-> stream is replicated)
        collector.emit(new Values(wrappedTuple,processId));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields(this.listOutFieldsStr));
    }
    private void readPropertiesFile(){
        this.logInfo("Reading properties file...");
        InputStream input = null;
        try {
            input = new FileInputStream(this.pathPropFile);
            // load a properties file
            this.prop.load(input);
        } catch (FileNotFoundException e) {
            this.logger.logError(e.toString());

        } catch (IOException e) {
            this.logger.logError(e.toString());
        }
        // retrieve configuration parameters
        this.listInFieldsStr =
        Arrays.asList(this.prop.getProperty("bolt.input.fields").split(","));
        this.listOutFieldsStr =
        Arrays.asList(this.prop.getProperty("bolt.output.fields").split(","));
        this.logInfo("Read done successfully");
    }
}
```

```java
package rt.deep.kafka;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class send messages into a Kafka topic
 */
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import org.apache.commons.io.IOUtils;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import org.json.simple.parser.ParseException;
import org.json.JSONArray;
import org.json.JSONObject;
import rt.deep.tuple.TupleApi;
import rt.deep.tuple.TupleApiSerializer;
import rt.deep.tuple.Field;
import rt.deep.tuple.Fields;
import rt.deep.tuple.CustomTupleImpl;

public class KafkaProducerUtil {

    // kafka producer properties from a load file
    private Properties kafkaProperties = new Properties();
    private Producer<String, TupleApi> producer;
    /**
     * Constructor
     */
    public KafkaProducerUtil (){
        this.kafkaProperties.put("bootstrap.servers", "localhost:9092");
        this.kafkaProperties.put("acks", "all");
        this.kafkaProperties.put("etries", 0);
        this.kafkaProperties.put("batch.size", 16384);
        this.kafkaProperties.put("linger.ms", 1);
        this.kafkaProperties.put("buffer.memory", 33554432);
        this.kafkaProperties.put("key.serializer",
        "org.apache.kafka.common.serialization.StringSerializer");
        //kafkaProperties.put("value.serializer",
        "org.apache.kafka.common.serialization.StringSerializer");
        this.kafkaProperties.put("value.serializer", TupleApiSerializer.class.getName());
        this.producer = new KafkaProducer<String, TupleApi>(this.kafkaProperties);
    }

    /**
     *
     * @param topic
     * @param key
     * @param msg
     */
    private void sendMsgTopic(String topic,String key,TupleApi msg){
        this.producer.send(new ProducerRecord<String, TupleApi>(topic, key, msg));

    }
    private void close(){
        this.producer.close();
    }
    /**
     *
     * @param file
     * @param topic
     * @throws ParseException
     */
    public void readJsonAndSendMsg(String file,String topic) throws ParseException{
        try {
            InputStream is = new FileInputStream(file);
            String jsonTxt = IOUtils.toString(is, "UTF-8");
            JSONArray jsonArray = new JSONArray(jsonTxt);
            for(int idxJson = 0; idxJson<jsonArray.length();idxJson++){
                JSONObject jsonObject =  jsonArray.getJSONObject(idxJson);
                JSONArray names = jsonObject.names();
                List<Field> lfield = new ArrayList<Field>();
                List<Object> lvalues = new ArrayList<Object>();
                for (int i = 0; i < names.length(); ++i) {
                    lfield.add(new Field(names.getString(i)));
                    lvalues.add(jsonObject.get(names.getString(i)).toString());
                }
                TupleApi tuple = new CustomTupleImpl(new Fields(lfield),lvalues);
                // send key,value into topic
                this.sendMsgTopic(topic,"id",tuple);
            }
            this.close();

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
package rt.deep.kafka;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class maps a tuple into Kafka mapper
 */
import org.apache.storm.kafka.bolt.mapper.TupleToKafkaMapper;
import org.apache.storm.tuple.Tuple;

public class NonDefaultFieldNameBasedTupleToKafkaMapper<K,V> implements
TupleToKafkaMapper<K, V> {
    /** serial ID */
    private static final long serialVersionUID = -1644874455499715486L;
    public static final String BOLT_KEY = "id";
    public static final String BOLT_MESSAGE = "winner";
    public String boltKeyField;
    public String boltMessageField;

    public NonDefaultFieldNameBasedTupleToKafkaMapper() {
        this(NonDefaultFieldNameBasedTupleToKafkaMapper.BOLT_KEY,
        NonDefaultFieldNameBasedTupleToKafkaMapper.BOLT_MESSAGE);
    }

    public NonDefaultFieldNameBasedTupleToKafkaMapper(String boltKeyField, String
    boltMessageField) {
        this.boltKeyField = boltKeyField;
        this.boltMessageField = boltMessageField;
    }

    @Override
    public K getKeyFromTuple(Tuple tuple) {
        //for backward compatibility, we return null when key is not present.
        return tuple.contains(this.boltKeyField) ? (K)
        tuple.getValueByField(this.boltKeyField) : null;
    }

    @Override
    public V getMessageFromTuple(Tuple tuple) {
        return (V) tuple.getValueByField(this.boltMessageField);
    }
}
```

```java
package rt.deep.kafka.spout;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class is used to Deserialize a mesage from a kafka spout
 */
import java.io.ByteArrayInputStream;
import java.io.ObjectInputStream;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.List;
import org.apache.storm.spout.Scheme;
import org.apache.storm.tuple.Fields;

public class SpoutDeserialize implements Scheme{
    /** serial ID */
    private static final long serialVersionUID = -8194872673337698705L;
    @Override
    public List<Object> deserialize(ByteBuffer ser) {
        List<Object> list = new ArrayList<Object>();
        try {
            byte[] arr = new byte[ser.remaining()];
            ser.get(arr);
            ByteArrayInputStream in = new ByteArrayInputStream(arr);
            ObjectInputStream is = new ObjectInputStream(in);
            list.add(is.readObject());
        } catch (Exception e) {

            e.printStackTrace();
        }
        return list;
    }
    @Override
    public Fields getOutputFields() {
        return new Fields ("id");
    }
}
```

```java
package rt.deep.log;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class used to keep trace and to give format to
 *  all input/ouput messages from the different components
 *  of the storm topology (spouts,bolts)
 */
import java.io.Serializable;
import java.sql.Timestamp;
import org.apache.log4j.Logger;

public class tracer implements Serializable {
    // default constructor
    public tracer(){
    }
    public String getBeginTrace() {
        return this.beginTrace;
    }
    public String getFinalTrace() {
        return this.finalTrace;
    }
    private final String beginTrace="BEGIN >>>";
    private final String logTool="LOG_ANALYSIS >>>";
    private final String finalTrace= "FINAL >>>";
    private static final long serialVersionUID = 1859463727395702470L;
    static Logger log = Logger.getLogger(tracer.class.getName());
    public void logInfo(String msg){
        tracer.log.info(this.logTool + new Timestamp(System.currentTimeMillis()).toString()
        + " INFO->"+msg);
    }
    public void logError(String msg){
        tracer.log.error(this.logTool + new Timestamp(System.currentTimeMillis()) + "
        ERR->"+msg);
    }

}
```

```java
package rt.deep.result;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class computes the majof vote of a data instance (TEST MODE)
 *  taking into account the predicted label from each classifier instance
 */
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;
import rt.deep.log.tracer;

public class ComputeMajorVote extends BaseBasicBolt {
    private String idName="computeMajorVote";
    /** serial ID */
    private static final long serialVersionUID = -3726297716111144181L;
    /** logger*/
    private tracer logger = new tracer();
    //properties file
    private Properties prop= new Properties();
    // path properties file
    private String pathPropFile;
    //list of fields read from properties file
    private List<String> listInFieldsStr;
    //list of fields read from properties file
    private List<String> listOutFieldsStr;
    //constructor
    public ComputeMajorVote(String pathPropertiesFile,int id){
        this.pathPropFile = pathPropertiesFile;
        this.readPropertiesFile();
        // custom idName
        this.idName = this.idName + "-"+String.valueOf(id);
    }
    @Override
    public void prepare(Map stormConf, TopologyContext context) {

    }
    // add begin trace : BOLT
    private void logOut(String msg){
        this.logger.logInfo(this.idName +" |BOLT OUT >>>"+msg);
    }
    // add begin trace : BOLT
    private void logIn(String msg){
        this.logger.logInfo(this.idName +" |BOLT IN >>>"+msg);
    }
    // add begin trace : SPOUT
    private void logInfo(String msg){
        this.logger.logInfo(this.idName +" |BOLT >>>"+msg);
    }
    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        if(tuple.size()>=2){
            // get idJoin
            String idJoin = (String)tuple.getValue(0);
            this.logIn("Computing Major Vote of idJoin="+idJoin);
            Integer maxLabel = this.predictedLabelCount(tuple);
            this.logOut("Emmiting Major Vote => idJoin " + idJoin + " MajorVote = "+
            maxLabel);
            collector.emit(new Values(idJoin,String.valueOf(maxLabel)));
        }
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields(this.listOutFieldsStr));
    }
    private void readPropertiesFile(){
        this.logInfo("Reading properties file...");
        InputStream input = null;
        try {
            input = new FileInputStream(this.pathPropFile);
            // load a properties file
            this.prop.load(input);
        } catch (FileNotFoundException e) {
            this.logger.logError(e.toString());

        } catch (IOException e) {
            this.logger.logError(e.toString());
        }
        // retrieve configuration parameters
        this.listInFieldsStr =
        Arrays.asList(this.prop.getProperty("bolt.input.fields").split(","));
        this.listOutFieldsStr =
        Arrays.asList(this.prop.getProperty("bolt.output.fields").split(","));
        this.logInfo("Read done successfully");
    }
    // return max label
    private int predictedLabelCount(Tuple tuple){
        Map<Integer, Integer> hm =this.counterPredictedLabels(tuple);
        int max=0;
        int label=0;
        for (Map.Entry<Integer, Integer> entry : hm.entrySet()) {
            if(max<entry.getValue()){
                max = entry.getValue();
                label = entry.getKey();
            }
            //System.out.println("Key = " + entry.getKey() + ", Value = " + entry.getValue());
        }
        return label;
    }

    private Map<Integer, Integer> counterPredictedLabels(Tuple tuple){
        Map<Integer, Integer> hm = new HashMap<Integer, Integer>();
        for(int i = 1; i<tuple.size();i++){
            Integer key = new Integer((Integer)tuple.getValue(i));
            if ( hm.containsKey(key) ) {
                int value = hm.get(key);
                hm.put(key, value + 1);
            } else {
                hm.put(key, 1);
            }
        }
        return hm;
    }
}
```

```
package rt.deep.synthetic.data.generation;

import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import org.apache.commons.math3.distribution.NormalDistribution;
import org.apache.commons.math3.distribution.UniformRealDistribution;

public class SyntheticDataGenerator {


    double [] mean;
    double [] std;
    double [] upper;
    double [] lower;
    List<NormalDistribution>  listNormalDistribution= new ArrayList<NormalDistribution>();
    List<UniformRealDistribution> listUniformRealDistribution = new
    ArrayList<UniformRealDistribution>();
    // default constructor generates random uniform distribution
    //and random normal distribution
    public SyntheticDataGenerator(Properties generalProperties){

        // read general properties
        String [] meanStr = generalProperties.getProperty("random.normal.mean").split(",");
        this.mean = new double [meanStr.length];
        for(int i = 0; i<meanStr.length;i++) {
            this.mean[i] = Double.valueOf(meanStr[i]);
        }

        String [] stdStr = generalProperties.getProperty("random.normal.std").split(",");
        this.std = new double [stdStr.length];
        for(int i = 0; i<stdStr.length;i++) {
            this.std[i] = Double.valueOf(stdStr[i]);
        }

        String [] uLowerStr =
        generalProperties.getProperty("random.uniform.lower").split(",");
        this.lower = new double [uLowerStr.length];
        for(int i = 0; i<uLowerStr.length;i++) {
            this.lower[i] = Double.valueOf(uLowerStr[i]);
        }

        String [] uUpperStr =
        generalProperties.getProperty("random.uniform.upper").split(",");
        this.upper = new double [uUpperStr.length];
        for(int i = 0; i<uUpperStr.length;i++) {
            this.upper[i] = Double.valueOf(uUpperStr[i]);
        }
        for(int i = 0;i<this.mean.length;i++){
            this.listNormalDistribution.add(new NormalDistribution(this.mean[i],this.std[i]));
        }
        for(int i =0; i<this.upper.length;i++){
            this.listUniformRealDistribution.add(new
            UniformRealDistribution(this.lower[i],this.upper[i]));
        }
    }
    public double getRandomSample(int idxSpout){
        double randomValue;
        switch (idxSpout) {
        case 1:
            randomValue = this.generateNormalDistributionSample(0);break;
        case 2:  randomValue = this.generateNormalDistributionSample(1);break;
        case 3:  randomValue = this.generateUniformDistributionSample(0);break;
        case 4:  randomValue = this.generateUniformDistributionSample(1);break;
        default: randomValue=0;break;
        }
        return randomValue;
    }
    public double generateNormalDistributionSample(int idx ){
        return this.listNormalDistribution.get(idx).sample();
    }

    public double generateUniformDistributionSample(int idx){
        return this.listUniformRealDistribution.get(idx).sample();
    }
}
```

```
package rt.deep.topology;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class create the rtDeep topology that will be used to submit
 *  in storm.
 */
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.TimeUnit;
import org.apache.storm.bolt.JoinBolt;
import org.apache.storm.kafka.BrokerHosts;
import org.apache.storm.kafka.KafkaSpout;
import org.apache.storm.kafka.ZkHosts;
import org.apache.storm.kafka.bolt.selector.DefaultTopicSelector;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.topology.base.BaseWindowedBolt;
import org.apache.storm.tuple.Fields;
import rt.deep.classifier.Classifier;
import rt.deep.filter.OperatorFilterSpout;
import rt.deep.filter.OperatorFilterTraningTest;
import rt.deep.kafka.NonDefaultFieldNameBasedTupleToKafkaMapper;
import rt.deep.result.ComputeMajorVote;
import org.apache.storm.kafka.bolt.KafkaBolt;
import rt.deep.utils.GeneralTools;
import rt.deep.utils.SelectOutputFieldJoin;

@SuppressWarnings("deprecation")
public class rtDeepTopology {
    Properties generalProperties;
    rtDeepTopology(Properties generalProp){
        this.generalProperties = generalProp;
    }
    public TopologyBuilder create() {

        // kafka input topics names
        String topicname1 = this.generalProperties.getProperty("kafkaspout1.topicname");
        String topicname2 = this.generalProperties.getProperty("kafkaspout2.topicname");
        String topicname3 = this.generalProperties.getProperty("kafkaspout3.topicname");
        String topicname4 = this.generalProperties.getProperty("kafkaspout4.topicname");
        // zookeeper host & port configuration
        String zookeeperHost = this.generalProperties.getProperty("zookeeper.host");
        String zookeeperPort = this.generalProperties.getProperty("zookeeper.port");
        String zkConnString = zookeeperHost+":"+zookeeperPort;
        // Brokerhost
        BrokerHosts hosts = new ZkHosts(zkConnString);
        // create input kafka spouts
        KafkaSpout kafkaSpoutOb1 = new
        KafkaSpout(GeneralTools.createSpoutConfig(hosts,topicname1));
        KafkaSpout kafkaSpoutOb2 = new
        KafkaSpout(GeneralTools.createSpoutConfig(hosts,topicname2));
        KafkaSpout kafkaSpoutOb3 = new
        KafkaSpout(GeneralTools.createSpoutConfig(hosts,topicname3));
        KafkaSpout kafkaSpoutOb4 = new
        KafkaSpout(GeneralTools.createSpoutConfig(hosts,topicname4));

        // Build topology
        // KafkaSpout input ---> Filter(if any) ->
        //                              Join ---> Classifier --> JoinResults
        //                                                       --> computeMajorVote --
        KafkaBolt output
        // consume message from kafka and print them on console
        final TopologyBuilder topologyBuilder = new TopologyBuilder();
        // kafka spout ids
        String kafkaSpout1Id = this.generalProperties.getProperty("kafkaspout1.id");
        String kafkaSpout2Id = this.generalProperties.getProperty("kafkaspout2.id");
        String kafkaSpout3Id = this.generalProperties.getProperty("kafkaspout3.id");
        String kafkaSpout4Id = this.generalProperties.getProperty("kafkaspout4.id");

        // Create KafkaSpout instance using Kafka configuration and add it to topology //input
        topologyBuilder.setSpout(kafkaSpout1Id, kafkaSpoutOb1, 1);
        topologyBuilder.setSpout(kafkaSpout2Id, kafkaSpoutOb2, 1);
        topologyBuilder.setSpout(kafkaSpout3Id, kafkaSpoutOb3, 1);
        topologyBuilder.setSpout(kafkaSpout4Id, kafkaSpoutOb4, 1);

        // get filder bolt ids
        String filterBoltSpout1Id =
        this.generalProperties.getProperty("bolt.filter.spout1.id");
        String filterBoltSpout2Id =
        this.generalProperties.getProperty("bolt.filter.spout2.id");
        String filterBoltSpout3Id =
        this.generalProperties.getProperty("bolt.filter.spout3.id");
        String filterBoltSpout4Id =
        this.generalProperties.getProperty("bolt.filter.spout4.id");

        // read configuration for topology bolts and spouts
        //filter items and filter group by id configuration files
        String boltFilterSpout1ConfigPath =
        this.generalProperties.getProperty("bolt1.filter.spout1.config.path");
        String boltFilterSpout2ConfigPath =
        this.generalProperties.getProperty("bolt2.filter.spout2.config.path");
        String boltFilterSpout3ConfigPath =
        this.generalProperties.getProperty("bolt3.filter.spout3.config.path");
        String boltFilterSpout4ConfigPath =
        this.generalProperties.getProperty("bolt4.filter.spout4.config.path");
        // add Filter Bolts to the topology
        //Shuffle grouping: Tuples are randomly distributed across the bolt's tasks in a way
        such that each bolt is guaranteed to get an equal number of tuples
        topologyBuilder.setBolt(filterBoltSpout1Id, new
        OperatorFilterSpout(boltFilterSpout1ConfigPath,1), 1).shuffleGrouping(kafkaSpout1Id);
        topologyBuilder.setBolt(filterBoltSpout2Id, new
        OperatorFilterSpout(boltFilterSpout2ConfigPath,2), 1).shuffleGrouping(kafkaSpout2Id);
        topologyBuilder.setBolt(filterBoltSpout3Id, new
        OperatorFilterSpout(boltFilterSpout3ConfigPath,3), 1).shuffleGrouping(kafkaSpout3Id);
        topologyBuilder.setBolt(filterBoltSpout4Id, new
        OperatorFilterSpout(boltFilterSpout4ConfigPath,4), 1).shuffleGrouping(kafkaSpout4Id);

        // get join id and output fields
        SelectOutputFieldJoin selectOutputFieldJoinFilter = new SelectOutputFieldJoin();
        String selectAllItemStreamFilter1 =
        selectOutputFieldJoinFilter.selectAllOutputFields(boltFilterSpout1ConfigPath,
        filterBoltSpout1Id);
        String selectAllItemStreamFilter2 =
        selectOutputFieldJoinFilter.selectAllOutputFields(boltFilterSpout2ConfigPath,
        filterBoltSpout2Id);
        String selectAllItemStreamFilter3 =
        selectOutputFieldJoinFilter.selectAllOutputFields(boltFilterSpout3ConfigPath,
        filterBoltSpout3Id);
        String selectAllItemStreamFilter4 =
        selectOutputFieldJoinFilter.selectAllOutputFields(boltFilterSpout4ConfigPath,
        filterBoltSpout4Id);
        String selectAllJoinFilter = filterBoltSpout1Id+":id,"
                +selectAllItemStreamFilter1+","
                +selectAllItemStreamFilter2+","
                +selectAllItemStreamFilter3+","
                +selectAllItemStreamFilter4;
        String joinInputId = this.generalProperties.getProperty("joiner.inputs.id");
        String joinInputKeyField =
        this.generalProperties.getProperty("joiner.inputs.key.field");
        //create Join to aggregate data from several observatories
        JoinBolt jbolt =  new JoinBolt(filterBoltSpout1Id, joinInputKeyField)
                .join     (filterBoltSpout2Id, joinInputKeyField,filterBoltSpout1Id)
                .join     (filterBoltSpout3Id, joinInputKeyField,filterBoltSpout2Id)
                .leftJoin (filterBoltSpout4Id, joinInputKeyField,filterBoltSpout3Id)
                .select   (selectAllJoinFilter)   // chose output fields
                .withTumblingWindow( new BaseWindowedBolt.Duration(10, TimeUnit.SECONDS)) ;

        // add join into the topology
        topologyBuilder.setBolt(joinInputId, jbolt, 1)
        .fieldsGrouping(filterBoltSpout1Id, new Fields(joinInputKeyField))
        .fieldsGrouping(filterBoltSpout2Id, new Fields(joinInputKeyField))
        .fieldsGrouping(filterBoltSpout3Id, new Fields(joinInputKeyField))
        .fieldsGrouping(filterBoltSpout4Id, new Fields(joinInputKeyField));

        // get filter training test ids
        String FilterTrainingTestId =
        this.generalProperties.getProperty("bolt.filter.training.and.test.id");
        String FilterTrainingTestConfigPath =
        this.generalProperties.getProperty("bolt.filter.training.and.test.config.path");

        // Global grouping: The entire stream goes to a single one of the bolt's tasks.
        Specifically, it goes to the task with the lowest id.
        // add filter training test to the topology
        topologyBuilder.setBolt(FilterTrainingTestId, new
        OperatorFilterTraningTest(FilterTrainingTestConfigPath,1),
        1).globalGrouping(joinInputId);

        // get filter bolt ids
        String classBolt1Id = this.generalProperties.getProperty("bolt1.classifier.id");
        String classBolt2Id = this.generalProperties.getProperty("bolt2.classifier.id");
        String classBolt3Id = this.generalProperties.getProperty("bolt3.classifier.id");
        String classBolt4Id = this.generalProperties.getProperty("bolt4.classifier.id");

        // Micro-Cluster Nearest Neighbors classifiers configuration files
        String bolt1ClassifierConfigPath =
        this.generalProperties.getProperty("bolt1.classifier.config.path");
        String bolt2ClassifierConfigPath =
        this.generalProperties.getProperty("bolt2.classifier.config.path");
        String bolt3ClassifierConfigPath =
        this.generalProperties.getProperty("bolt3.classifier.config.path");
        String bolt4ClassifierConfigPath =
        this.generalProperties.getProperty("bolt4.classifier.config.path");

        String
        classLabelList=this.generalProperties.getProperty("classifier.class.label.list");
        String [] splitList = classLabelList.split(",");
        List<Integer> listClass = new ArrayList<Integer>();
        for(int i =0; i <splitList.length;i++){
            listClass.add(Integer.valueOf(splitList[i]));
        }
        //  all grouping : the stream is replicated across all the bolt's tasks. Use this
        grouping with care.
        // add classifier bolts into the topology
        topologyBuilder.setBolt(classBolt1Id, new
        Classifier(bolt1ClassifierConfigPath,listClass,1),
        1).allGrouping(FilterTrainingTestId);
        topologyBuilder.setBolt(classBolt2Id, new
        Classifier(bolt2ClassifierConfigPath,listClass,2),
        1).allGrouping(FilterTrainingTestId);
        topologyBuilder.setBolt(classBolt3Id, new
        Classifier(bolt3ClassifierConfigPath,listClass,3),
        1).allGrouping(FilterTrainingTestId);
        topologyBuilder.setBolt(classBolt4Id, new
        Classifier(bolt4ClassifierConfigPath,listClass,4),
        1).allGrouping(FilterTrainingTestId);

        // get join id and output fields

        String joinClassifierId = this.generalProperties.getProperty("joiner.classifier.id");
```

```
        String joinClassifierKeyField =
        this.generalProperties.getProperty("joiner.classifier.key.field");
        // get join id and output fields
        SelectOutputFieldJoin selectOutputFieldJoinClassifier = new SelectOutputFieldJoin();
        String selectAllItemStreamClass1 =
        selectOutputFieldJoinClassifier.selectAllOutputFields(bolt1ClassifierConfigPath,
        classBolt1Id);
        String selectAllItemStreamClass2 =
        selectOutputFieldJoinClassifier.selectAllOutputFields(bolt2ClassifierConfigPath,
        classBolt2Id);
        String selectAllItemStreamClass3 =
        selectOutputFieldJoinClassifier.selectAllOutputFields(bolt3ClassifierConfigPath,
        classBolt3Id);
        String selectAllItemStreamClass4 =
        selectOutputFieldJoinClassifier.selectAllOutputFields(bolt4ClassifierConfigPath,
        classBolt4Id);
        String selectAllJoinClassifier = classBolt1Id+":id,"
                +selectAllItemStreamClass1+","
                +selectAllItemStreamClass2+","
                +selectAllItemStreamClass3+","
                +selectAllItemStreamClass4;

        //create Join to aggregate data from several observatories
        JoinBolt jboltClassifier =  new JoinBolt(classBolt1Id, joinClassifierKeyField)
                .join     (classBolt2Id, joinClassifierKeyField,  classBolt1Id)
                .join     (classBolt3Id, joinClassifierKeyField,    classBolt2Id)
                .leftJoin (classBolt4Id, joinClassifierKeyField,    classBolt3Id)
                .select   (selectAllJoinClassifier)   // chose output fields
                .withTumblingWindow( new BaseWindowedBolt.Duration(10, TimeUnit.SECONDS)) ;

        // add join to aggregate the prediction of each classifiers
        topologyBuilder.setBolt(joinClassifierId, jboltClassifier, 1)
        .fieldsGrouping(classBolt1Id, new Fields(joinClassifierKeyField))
        .fieldsGrouping(classBolt2Id, new Fields(joinClassifierKeyField))
        .fieldsGrouping(classBolt3Id, new Fields(joinClassifierKeyField))
        .fieldsGrouping(classBolt4Id, new Fields(joinClassifierKeyField));

        // get compute major bolt id
        String computeMajorVoteBoltId =
        this.generalProperties.getProperty("bolt.compute.major.vote.id");
        String boltComputeMajorVoteConfigPath =
        this.generalProperties.getProperty("bolt.compute.major.vote.config.path");
        // add compute major vote into the topology
        topologyBuilder.setBolt(computeMajorVoteBoltId, new
        ComputeMajorVote(boltComputeMajorVoteConfigPath,1),
        1).globalGrouping(joinClassifierId);

        // load kafka producer properties from file
        Properties kafkaProducerProperties = new Properties();
        try {
            InputStream inputkafka = new
            FileInputStream(this.generalProperties.getProperty("kafka.output.config.path"));
            kafkaProducerProperties.load(inputkafka);
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        @SuppressWarnings({ "unchecked", "rawtypes" })
        String kafkaOutputId = this.generalProperties.getProperty("kafka.output.id");
        String kafkaOutputTopicname =
        this.generalProperties.getProperty("kafka.ouput.topicname");

        @SuppressWarnings("rawtypes")
        KafkaBolt kafkaBolt = new KafkaBolt()
        .withProducerProperties(kafkaProducerProperties)
        .withTopicSelector(new DefaultTopicSelector(kafkaOutputTopicname))
        .withTupleToKafkaMapper(new NonDefaultFieldNameBasedTupleToKafkaMapper());
        topologyBuilder.setBolt(kafkaOutputId, kafkaBolt,
        1).shuffleGrouping(computeMajorVoteBoltId);
        return topologyBuilder;
    }
}
```

```java
package rt.deep.tuple;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class is used to generate Custom tuple that will be passed
 *  within the topology and implements the interface defined in TupleApi
 */
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Properties;

import com.google.common.collect.Lists;

public class CustomTupleImpl implements TupleApi {
    /** general serial UID */
    private static final long serialVersionUID = 2644853349762681098L;
    private Properties properties;
    private final Fields schema;
    private String tupleName;
    private final List<Object> values;

    public CustomTupleImpl()
    {
        this.values = new ArrayList();
        this.schema = new Fields();
        this.properties = new Properties();
    }
    public CustomTupleImpl(Field field, Object value)
    {
        this(new Fields(new Field[] { field }), Lists.newArrayList(new Object[] { value }));
    }

    public CustomTupleImpl(Fields schema, List<Object> values)
    {
        this.values = values;
        this.schema = schema;
        if (values.size() != schema.size()) {
            throw new IllegalArgumentException("Tuple created with wrong number  of fields.
            Expected " + schema.size() + " fields but got " + values.size() + " fields");
        }
        this.properties = new Properties();
    }

    public CustomTupleImpl(String tupleName, Field field, Object value)
    {
        this(new Fields(new Field[] { field }), Lists.newArrayList(new Object[] { value }));
        this.tupleName = tupleName;
    }

    public CustomTupleImpl(String name, Fields schema, List<Object> values)
    {
        this.values = values;
        this.schema = schema;
        this.tupleName = name;
        if (values.size() != schema.size()) {
            throw new IllegalArgumentException("Tuple created with wrong number  of fields.
            Expected " + schema.size() + " fields but got " + values.size() + " fields");
        }
        this.properties = new Properties();
    }

    public CustomTupleImpl(String fieldName, Object value)
    {
        this(new Field(fieldName), value);
    }
    public CustomTupleImpl(String tupleName, String fieldName, Object value)
    {
        this(new Field(fieldName), value);
        this.tupleName = tupleName;
    }

    public static final TupleApi valueOf(Object obj)
    {
        TupleApi tuple;
        if (null == obj) {
            throw new IllegalArgumentException("Given object is null.");
        }
        try
        {
            tuple = (TupleApi)obj;
        }
        catch (ClassCastException e)
        {
            throw new IllegalArgumentException("Given object cannot be casted into
            ITupleApi.");
        }
        return tuple;
    }

    @Override
    public final void addField(String pathname, Object field)
    {
        Path path = Paths.get(pathname, new String[0]);
        CustomTupleImpl tuple = this;
        for (int i = 0; i < (path.getNameCount() - 1); i++)
        {
            String fieldName = path.getName(i).toString();
            tuple = (CustomTupleImpl)tuple.getValueByField(fieldName);
        }
        String fieldName = path.getName(path.getNameCount() - 1).toString();
        tuple.getValues().add(this.fieldIndex(fieldName), field);
    }

    @Override
    public final boolean contains(String field)
    {
        return this.getFields().contains(field);
    }

    @Override
    public final boolean equals(Object other)
    {
        if (other == this) {
            return true;
        }
        if ((other instanceof CustomTupleImpl))
        {
            CustomTupleImpl otherTuple = (CustomTupleImpl)other;
            if
            (!this.getFields().get(0).getName().equals(otherTuple.getFields().get(0).getName()
            )) {
                return false;
            }
            if (!this.getValue(0).equals(otherTuple.getValue(0))) {
                return false;
            }
            return true;
        }
        return false;
    }

    @Override
```

```java
    public final int fieldIndex(String field)
    {
        return this.getFields().fieldIndex(field);
    }

    @Override
    public final Object find(String pathName)
    {
        Path path = Paths.get(pathName, new String[0]);
        CustomTupleImpl tuple = this;
        for (int i = 0; i < (path.getNameCount() - 1); i++)
        {
            String fieldName = path.getName(i).toString();
            tuple = (CustomTupleImpl)tuple.getValueByField(fieldName);
        }
        String fieldName = path.getName(path.getNameCount() - 1).toString();
        return tuple.getValueByField(fieldName);
    }

    @Override
    public final Field getField(String field)
    {
        return this.schema.get(this.fieldIndex(field));
    }

    @Override
    public final Fields getFields()
    {
        return this.schema;
    }

    @Override
    public final Properties getProperties()
    {
        return this.properties;
    }

    @Override
    public final String getTupleName()
    {
        return this.tupleName;
    }

    @Override
    public final Object getValue(int i)
    {
        return this.values.get(i);
    }

    @Override
    public final Object getValueByField(String field)
    {
        return this.values.get(this.fieldIndex(field));
    }

    @Override
    public final List<Object> getValues()
    {
        return this.values;
    }

    @Override
    public final int hashCode()
    {
        return this.getFields().get(0).getName().hashCode() + this.getValue(0).hashCode();
    }

    @Override
    public final List<Object> select(Fields selector)
    {
        return this.getFields().select(selector, this.values);
    }

    @Override
    public final void setName(String name)
    {
        this.tupleName = name;
    }

    @Override
    public final void setProperties(Properties properties)
    {
        this.properties = properties;
    }

    @Override
    public final int size()
    {
        return this.values.size();
    }

    @Override
    public final String toString()
    {
        StringBuffer out = new StringBuffer();
        out.append("{");
        for (int i = 0; i < this.schema.size(); i++)
        {
            Field field = this.schema.get(i);
            Object value = this.values.get(i);
            if (value != null)
            {
                if ((value instanceof List)) {
                    value = ((List)value).subList(0, Math.min(((List)value).size(), 1000));
                }
                if (value.getClass().isArray()) {
                    value = this.arraytoString(value);
                }
            }
            out.append(field.getName() + " -> " + value + ";");
        }
        out.append("}");
        return out.toString();
    }

    private String arraytoString(Object obj)
    {
        StringBuffer buffer = new StringBuffer();
        if (!(obj instanceof Object[]))
        {
            int size = 0;
            if ((obj instanceof int[])) {
                size = this.intArraytoString(obj, buffer, size);
            }
            if ((obj instanceof char[])) {
                size = this.charArraytoString(obj, buffer, size);
            }
            if ((obj instanceof float[])) {
                size = this.floatArraytoString(obj, buffer, size);
            }
            if ((obj instanceof double[])) {
                size = this.doubleArraytoString(obj, buffer, size);
            }
            if ((obj instanceof short[])) {
                size = this.shortArraytoString(obj, buffer, size);
            }
        }
        else
        {
            buffer.append(Arrays.asList((Object[])obj));
        }
        return buffer.toString();
    }

    private int charArraytoString(Object obj, StringBuffer buffer, int size)
    {
        int aSize = size;
        for (char c : (char[])obj)
        {
            buffer.append(c + ", ");
            if (aSize++ >= 1000) {
                break;
            }
        }
        return aSize;
    }

    private int doubleArraytoString(Object obj, StringBuffer buffer, int size)
    {
        int aSize = size;
        for (double c : (double[])obj)
        {
            buffer.append(c + ", ");
            if (aSize++ >= 1000) {
                break;
            }
        }
        return aSize;
    }

    private int floatArraytoString(Object obj, StringBuffer buffer, int size)
    {
        int aSize = size;
        for (float c : (float[])obj)
        {
            buffer.append(c + ", ");
            if (aSize++ >= 1000) {
                break;
            }
        }
        return aSize;
    }

    private int intArraytoString(Object obj, StringBuffer buffer, int size)
    {
        int aSize = size;
        for (int i : (int[])obj)
        {
            buffer.append(i + ", ");
            if (aSize++ >= 1000) {
                break;
            }
        }
        return aSize;
    }

    private int shortArraytoString(Object obj, StringBuffer buffer, int size)
    {
        int aSize = size;
        for (short c : (short[])obj)
        {
            buffer.append(c + ", ");
            if (aSize++ >= 1000) {
                break;
            }
        }
        return aSize;
    }
}
```

```java
package rt.deep.tuple;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class define some data type used in Fields Class
 */
import java.io.Serializable;
import java.util.Arrays;

public class Datatype implements Serializable{
    // serial UID
    private static final long serialVersionUID = -5913440578132889348L;
    private final BaseType baseType;
    private String[] enums;
    private Fields fields;

    public static enum BaseType
    {
        DT_BOOLEAN(Boolean.class),  DT_BYTE(Byte.class),  DT_CHAR(Character.class),
        DT_DOUBLE(Double.class),  DT_ENUM1(String.class),  DT_ENUM2(String.class),
        DT_ENUM4(String.class),  DT_FLOAT(Float.class),  DT_INT(Integer.class),
        DT_LONG(Long.class),  DT_OBJECT(Object.class),  DT_SHORT(Short.class),
        DT_STRING(String.class),  TUPLE(Object.class);

        private final Class<?> clazz;

        private BaseType(Class<?> clazz)
        {
            this.clazz = clazz;
        }

        public Class<?> getClazz()
        {
            return this.clazz;
        }
    }

    public Datatype()
    {
        this.baseType = BaseType.DT_BOOLEAN;
    }

    public Datatype(BaseType baseType)
    {
        if (baseType == BaseType.TUPLE) {
            throw new RuntimeException("Tuple should be initialized with fields (shema)");
        }
        this.baseType = baseType;
    }

    public Datatype(Fields fields)
    {
        this.baseType = BaseType.TUPLE;
        this.fields = fields;
    }

    public final BaseType getBaseType()
    {
        return this.baseType;
    }

    public final String[] getEnums()
    {
        return this.enums;
    }

    public final Fields getFields()
    {
        return this.fields;
    }

    public final void setEnums(String[] enums)
    {
        this.enums = enums;
    }

    public final void setFields(Fields fields)
    {
        this.fields = fields;
    }

    @Override
    public final String toString()
    {
        return "Datatype [baseType=" + this.baseType + ", enums=" +
        Arrays.toString(this.enums) + ", fields=" + this.fields + "]";
    }
}
```

```java
package rt.deep.tuple;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class defines a field within a tuple
 */
import java.io.Serializable;

public class Field implements Serializable{
    /** serial UID*/
    private static final long serialVersionUID = 8415506891188420654L;
    private boolean isArray;
    private boolean isUnsigned;
    private String name;
    private Datatype type;

    public Field() {}

    public Field(String name)
    {
        this(name, new Datatype(Datatype.BaseType.DT_OBJECT), false);
    }

    public Field(String name, Datatype type)
    {
        this(name, type, false);
    }

    public Field(String name, Datatype type, boolean isArray)
    {
        if ("boolean".equals(name)) {
            this.name = (name + "__________");
        } else {
            this.name = name;
        }
        this.type = type;
        this.isArray = isArray;
    }

    public Field(String name, Datatype type, boolean isArray, boolean isUnsigned)
    {
        if ("boolean".equals(name)) {
            this.name = (name + "__________");
        } else {
            this.name = name;
        }
        this.type = type;
        this.isArray = isArray;
        this.isUnsigned = isUnsigned;
    }

    public final String getName()
    {
        return this.name;
    }

    public final Datatype getType()
    {
        return this.type;
    }

    public final boolean isArray()
    {
        return this.isArray;
    }

    public final boolean isUnsigned()
    {
        return this.isUnsigned;
    }

    public final void setArray(boolean array)
    {
        this.isArray = array;
    }

    public final void setName(String name)
    {
        this.name = name;
    }

    public final void setType(Datatype type)
    {
        this.type = type;
    }

    @Override
    public final String toString()
    {
        return "Field [name=" + this.name + ", type=" + this.type + ", array=" +
        this.isArray + "]";
    }
}
```

```java
package rt.deep.tuple;

/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class defines a list of Fields within a tuple
 *  and implements Iterable Field
 */
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

public class Fields implements Iterable<Field>, Serializable{
    /** serial UID*/
    private static final long serialVersionUID = -8154172480131062720L;
    private final List<Field> fieldsList;
    private final Map<String, Integer> mapIndex = new HashMap();

    public Fields()
    {
        this.fieldsList = new ArrayList();
    }

    public Fields(Field... fields)
    {
        this(Arrays.asList(fields));
    }

    public Fields(List<Field> fields)
    {
        this.fieldsList = new ArrayList(fields.size());
        for (Field field : fields)
        {
            if (this.fieldsList.contains(field)) {
                throw new IllegalArgumentException(String.format("duplicate field '%s'", new
                Object[] { field }));
            }
            this.fieldsList.add(field);
        }
        this.index();
    }

    public Fields(String... fields)
    {
        this.fieldsList = new ArrayList(fields.length);
        for (String fieldStr : fields)
        {
            Field field = new Field(fieldStr);
            if (this.fieldsList.contains(field)) {
                throw new IllegalArgumentException(String.format("duplicate field '%s'", new
                Object[] { field }));
            }
            this.fieldsList.add(field);
        }
        this.index();
    }

    public final boolean contains(String field)
    {
        return this.mapIndex.containsKey(field);
    }

    public final int fieldIndex(String field)
    {
        Integer ret = this.mapIndex.get(field);
        if (ret == null) {
            throw new IllegalArgumentException(field + " does not exist");
        }
        return ret.intValue();
    }

    public final Field get(int index)
    {
        return this.fieldsList.get(index);
    }

    @Override
    public final Iterator<Field> iterator()
    {
        return this.fieldsList.iterator();
    }

    public final List<Object> select(Fields selector, List<Object> tuple)
    {
        List<Object> ret = new ArrayList(selector.size());
        for (Field s : selector) {
            ret.add(tuple.get(this.mapIndex.get(s).intValue()));
        }
        return ret;
    }

    public final int size()
    {
        return this.fieldsList.size();
    }

    public final List<Field> toList()
    {
        return new ArrayList(this.fieldsList);
    }

    @Override
    public final String toString()
    {
        return this.fieldsList.toString();
    }

    private void index()
    {
        for (int i = 0; i < this.fieldsList.size(); i++) {
            this.mapIndex.put(this.fieldsList.get(i).getName(), Integer.valueOf(i));
        }
    }
}
```

```java
package rt.deep.tuple;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This interface is defined for CustomTupleImpl
 */
import java.io.Serializable;
import java.util.List;
import java.util.Properties;

public abstract interface TupleApi extends Serializable{

    public abstract void addField(String paramString, Object paramObject);

    public abstract boolean contains(String paramString);

    public abstract int fieldIndex(String paramString);

    public abstract Object find(String paramString);

    public abstract Field getField(String paramString);

    public abstract Fields getFields();

    public abstract Properties getProperties();

    public abstract String getTupleName();

    public abstract Object getValue(int paramInt);

    public abstract Object getValueByField(String paramString);

    public abstract List<Object> getValues();

    public abstract List<Object> select(Fields paramFields);

    public abstract void setName(String paramString);

    public abstract void setProperties(Properties paramProperties);

    public abstract int size();
}
```

```java
package rt.deep.tuple;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class DeSerializes a TupleApi
 */
import java.util.Map;
import org.apache.kafka.common.serialization.Deserializer;
import com.fasterxml.jackson.databind.ObjectMapper;
import rt.deep.tuple.TupleApi;
@SuppressWarnings("rawtypes")
public class TupleApiDeSerializer implements Deserializer {
    @Override
    public TupleApi deserialize(String arg0, byte[] arg1) {
        ObjectMapper mapper = new ObjectMapper();
        TupleApi tupleapi = null;
        try {
            tupleapi = mapper.readValue(arg1, TupleApi.class);
        } catch (Exception e) {

            e.printStackTrace();
        }
        return tupleapi;
    }
    @Override
    public void configure(Map arg0, boolean arg1) {
        // TODO Auto-generated method stub

    }
    @Override public void close() {

    }
}
```

```java
package rt.deep.tuple;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class Serializes a TupleApi
 */
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.Map;
import org.apache.kafka.common.serialization.Serializer;
@SuppressWarnings("rawtypes")

public class TupleApiSerializer implements Serializer {
    @Override public byte[] serialize(String arg0, Object arg1) {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        try {
            ObjectOutputStream os = new ObjectOutputStream(bos);
            os.writeObject(arg1);
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        return bos.toByteArray();
    }

    @Override public void close() {

    }
    @Override
    public void configure(Map arg0, boolean arg1) {
        // TODO Auto-generated method stub

    }
}
```

```java
package rt.deep.utils;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class contains several methods that are useful within the topology
 */
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import org.apache.storm.kafka.BrokerHosts;
import org.apache.storm.kafka.SpoutConfig;
import org.apache.storm.spout.SchemeAsMultiScheme;
import org.apache.storm.tuple.Tuple;
import rt.deep.tuple.TupleApi;
import rt.deep.tuple.Fields;
import rt.deep.tuple.Field;
import rt.deep.tuple.CustomTupleImpl;
import rt.deep.constants.RtDeepContants;
import rt.deep.kafka.spout.SpoutDeserialize;
public class GeneralTools {
    public static SpoutConfig createSpoutConfig(BrokerHosts host, String topicname){
        // create spout config
        SpoutConfig spoutConfig = new SpoutConfig(host, topicname, "/" + topicname,
        UUID.randomUUID().toString());
        spoutConfig.scheme = new SchemeAsMultiScheme(new SpoutDeserialize());
        spoutConfig.ignoreZkOffsets = true;
        spoutConfig.startOffsetTime = -2;
        return spoutConfig;
    }
    public static TupleApi WrappTuples(Tuple tuple){
        String idJoinTuple="";
        List<TupleApi> inputTuples = new ArrayList<TupleApi>();
        // get id and TupleApis
        for(int i = 0; i<tuple.size();i++){
            if(i==0){ // get id
                idJoinTuple =(String)tuple.getValue(i);
            }
            else{ // get tuples
                inputTuples.add((TupleApi)tuple.getValue(i));
            }
        }
        // wrap all tuples into one tuple
        List<Field> lfields = new ArrayList<Field>();
        List<Object> lvalues = new ArrayList<Object>();
        lfields.add(new Field(RtDeepContants.ID_JOIN_FIELD));
        lfields.add(new Field(RtDeepContants.LIST_TUPLE_FIELD));
        lvalues.add(idJoinTuple);
        lvalues.add(inputTuples);
        TupleApi wrappedTuple= new CustomTupleImpl(new Fields(lfields),lvalues);
        return wrappedTuple;
    }
}
```

```java
package rt.deep.utils;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;
import java.util.List;
import java.util.Properties;

import rt.deep.log.tracer;
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *  This class Selects OutputFields from a bolt (Except Identifier)
 */
public class SelectOutputFieldJoin {
    private tracer logger = new tracer();
    //list of fields read from properties file
    private List<String> listOutFieldsStr;
    private void readPropertiesFile(String fileName){
        Properties prop = new Properties();
        this.logger.logInfo("Reading properties file...");
        InputStream input = null;
        try {
            input = new FileInputStream(fileName);
            // load a properties file
            prop.load(input);
        } catch (FileNotFoundException e) {
            this.logger.logError(e.toString());

        } catch (IOException e) {
            this.logger.logError(e.toString());
        }
        // retrieve configuration parameters
        this.listOutFieldsStr =
        Arrays.asList(prop.getProperty("bolt.output.fields").split(","));
        this.logger.logInfo("Read done.");
    }
    public String selectAllOutputFields(String fileName,String idComponent){
        this.readPropertiesFile(fileName);
        String streamItems="";
        for(int i = 0; i<this.listOutFieldsStr.size();i++){
            if(!this.listOutFieldsStr.get(i).equals("id")){
                streamItems +=idComponent+":"+this.listOutFieldsStr.get(i);
                if(i!=(this.listOutFieldsStr.size()-1)){
                    streamItems+=",";
                }
            }
        }
        return streamItems;
    }
}
```

```
package rt.deep.synthetic.data;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

import rt.deep.synthetic.data.generation.SyntheticDataGenerator;

public class GeneratorDataTest {
    // specific parameters used to created successfully the statistical distributions
    public static final String GENERAL_PROPERTIES_CLASS_ZERO_FILE =
    "src/test/resources/randomGenerator/randomGeneratorClassLabel0.properties";
    public static final String GENERAL_PROPERTIES_CLASS_ONE_FILE =
    "src/test/resources/randomGenerator/randomGeneratorClassLabel1.properties";
    // number of spout
    public static int TOTAL_SPOUT = 4;
    // number of parameters per spout
    public static int NUM_FIELD_SPOUT1 = 4;
    public static int NUM_FIELD_SPOUT2 = 2;
    public static int NUM_FIELD_SPOUT3 = 1;
    public static int NUM_FIELD_SPOUT4 = 5;
    // number of training set
    public static int NUM_TRAINING_SET = 1000;
    // number of test set
    public static int NUM_TEST_SET = 100;
    @SuppressWarnings("unchecked")
    // main method which generates a json file containing two possible scenarios: normal or
    tsunami as it is specified in detail in the report
    public static void main(String[] args) {
        List<Integer> listNumItemSpout = new ArrayList<Integer>();
        listNumItemSpout.add(GeneratorDataTest.NUM_FIELD_SPOUT1);
        listNumItemSpout.add(GeneratorDataTest.NUM_FIELD_SPOUT2);
        listNumItemSpout.add(GeneratorDataTest.NUM_FIELD_SPOUT3);
        listNumItemSpout.add(GeneratorDataTest.NUM_FIELD_SPOUT4);
        // compute offset
        List<Integer> spoutOffset = new ArrayList<Integer>();
        spoutOffset.add(0);
        spoutOffset.add(GeneratorDataTest.NUM_FIELD_SPOUT1);

        spoutOffset.add(GeneratorDataTest.NUM_FIELD_SPOUT1+GeneratorDataTest.NUM_FIELD_SPOUT2)
        ;

        spoutOffset.add(GeneratorDataTest.NUM_FIELD_SPOUT1+GeneratorDataTest.NUM_FIELD_SPOUT2+
        GeneratorDataTest.NUM_FIELD_SPOUT3);
        // open both properties
        Properties generalPropertiesZeroClass = new Properties();
        Properties generalPropertiesOneClass = new Properties();
        try {
            InputStream inputZero = null;
            inputZero = new
            FileInputStream(GeneratorDataTest.GENERAL_PROPERTIES_CLASS_ZERO_FILE);
            generalPropertiesZeroClass.load(inputZero);
            InputStream inputOne = null;
            inputOne = new
            FileInputStream(GeneratorDataTest.GENERAL_PROPERTIES_CLASS_ONE_FILE);
            generalPropertiesOneClass.load(inputOne);
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        // Synthetic Data generator used to generate random numbers following a distribution
        SyntheticDataGenerator dataGeneratorZeroClass = new
        SyntheticDataGenerator(generalPropertiesZeroClass);
        SyntheticDataGenerator dataGeneratorOneClass= new
        SyntheticDataGenerator(generalPropertiesOneClass);
        double [][] matrixData = new double[GeneratorDataTest.NUM_TRAINING_SET][13];
        // for each spout
        for(int idxSpout = 1; idxSpout<=GeneratorDataTest.TOTAL_SPOUT;idxSpout++){
            File file = new
            File("src/test/resources/inputJSONKafkaTopic/testSpout"+String.valueOf(idxSpout)+"
            .json");
            JSONArray listJsonArray = new JSONArray();
            for(int idx = 0;idx<GeneratorDataTest.NUM_TRAINING_SET;idx++){
                JSONObject obj = new JSONObject();
                obj.put("id", idx+1);
                for(int idxItem = 0; idxItem<listNumItemSpout.get(idxSpout-1);idxItem++){
                    // zero class generation
                    if(idx<(GeneratorDataTest.NUM_TRAINING_SET/2)){
                        // first two spout follow normal distribution
                        double tmp = dataGeneratorZeroClass.getRandomSample(idxSpout);
                        matrixData[idx][spoutOffset.get(idxSpout-1)+idxItem]=tmp;
                        obj.put("param"+String.valueOf(idxItem), tmp );
                    }
                    // one class generation
                    else{
                        double tmp = dataGeneratorOneClass.getRandomSample(idxSpout);
                        matrixData[idx][spoutOffset.get(idxSpout-1)+idxItem]=tmp;
                        obj.put("param"+String.valueOf(idxItem), tmp );
                    }
                }
                if(idx<(GeneratorDataTest.NUM_TRAINING_SET/2)) {
                    matrixData[idx][12]=0.;
                    obj.put("classlabel", "0");
                } else {
                    matrixData[idx][12]=1.;
                    obj.put("classlabel", "1");
                }
                listJsonArray.add(obj);
            } // end spout items
            for(int i =
            GeneratorDataTest.NUM_TRAINING_SET;i<(GeneratorDataTest.NUM_TRAINING_SET+Generator
            DataTest.NUM_TEST_SET);i++){
                JSONObject obj = new JSONObject();
                obj.put("id", i+1);
                for(int idxItem = 0; idxItem<listNumItemSpout.get(idxSpout-1);idxItem++){
                    // zero class generation

                    if(i<(GeneratorDataTest.NUM_TRAINING_SET+(GeneratorDataTest.NUM_TEST_SET/2
                    ))){
                        // first two spout follow normal distribution
                        obj.put("param"+String.valueOf(idxItem),
                        dataGeneratorZeroClass.getRandomSample(idxSpout));
                    }
                    // one class generation
                    else{
                        obj.put("param"+String.valueOf(idxItem),
                        dataGeneratorOneClass.getRandomSample(idxSpout));
                    }
                } // end spout items
                listJsonArray.add(obj);
            }
            // write JSON file to be used at the beggining of the topology
            FileWriter fileWriter;
            try {
                //if(!file.exists()) {
                file.createNewFile();
                //}
                fileWriter = new FileWriter(file);
                System.out.println("Writing JSON object to file");
                fileWriter.write(listJsonArray.toJSONString());
                fileWriter.flush();
                fileWriter.close();
                System.out.println("Successfully Copied JSON Object to File...");
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
        // write a CSV file to plot input using MDS algorithm
        FileWriter fileWriterCSV;
        File filecsv = new File("src/test/resources/inputJSONKafkaTopic/dataSynthetic.csv");
        try {
            filecsv.createNewFile();
            fileWriterCSV = new FileWriter(filecsv);
            System.out.println("Writing matrix into CSV file");
            for(int line =0; line<GeneratorDataTest.NUM_TRAINING_SET;line++){
                String writeLine="";
                // convert into String value (separated by commas)
                for(int col=0; col<12;col++){
                    writeLine += String.valueOf(matrixData[line][col])+ ",";
                }
                // last line without comma
                writeLine += String.valueOf(matrixData[line][12])+"\n";
                fileWriterCSV.write(writeLine);
                fileWriterCSV.flush();
            }

            fileWriterCSV.close();
            System.out.println("Successfully CSV File created...");
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

```
/**
 *  Author: Roberto Robledo
 *  Version : 1.0
 *
 *  Topology Test
 *  This test creates runs a local topology test
 *
 *  In this prototype 4 spouts are created as input and they waiting messages from different
 topics.
 *  Therefore, a join operation is done to merge all inputs.
 *  Then, depending two modes are possible:
 *  - Training
 *  - Test
 *  When training mode is enable only one classifier instance processes the input stream
 *  Nonetheless, in test mode all the classifiers MC-NN(Micro-Cluster Nearest Neighbors)
 instances process the input stream
 *  A join operation is done again to merge all the predicted label of each classifiers
 *  Finally, a bolt is on charge of computing the major vote and it is pushed into a kafka
 ouput topics
 *
 *  TOPOLOGY SPOUT and BOLTS ELEMENTS :
 *  [SPOUT] [FILTER] [JOIN-INPUTS] [FILTER-TRAIN-TEST]
 *  -[MC-NN]-[JOIN-CLASSIFIER] [COMPUTE-MAJOR-VOTE] [KAFKA-OUTPUT]
 */

package rt.deep.topology;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
import org.apache.log4j.Logger;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.json.simple.parser.ParseException;
import rt.deep.log.tracer;
import rt.deep.tuple.TupleApi;

import rt.deep.kafka.KafkaProducerUtil;

public class topologyTest {
    private static final String GENERAL_PROPERTIES_FILE =
    "src/test/resources/topology.properties";
    private static Logger log = Logger.getLogger(tracer.class.getName());
    // MAIN TEST
    public static void main(String[] args) {
        // read general properties
        Properties generalProperties = new Properties();
        try {
            InputStream input = null;
            input = new FileInputStream(topologyTest.GENERAL_PROPERTIES_FILE);
            generalProperties.load(input);
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        boolean pushMessagesIntoTopic =
        Boolean.valueOf(generalProperties.getProperty("topology.fill.kafka.topics"));
        // kafka input topics names
        String topicname1 = generalProperties.getProperty("kafkaspout1.topicname");
        String topicname2 = generalProperties.getProperty("kafkaspout2.topicname");
        String topicname3 = generalProperties.getProperty("kafkaspout3.topicname");
        String topicname4 = generalProperties.getProperty("kafkaspout4.topicname");
        // push dummy values into Kafka topics before topology execution
        if(pushMessagesIntoTopic){
            String jsonFileSpout1 =
            generalProperties.getProperty("topology.kafka.json.file.spout1");
            String jsonFileSpout2 =
            generalProperties.getProperty("topology.kafka.json.file.spout2");
            String jsonFileSpout3 =
            generalProperties.getProperty("topology.kafka.json.file.spout3");
            String jsonFileSpout4 =
            generalProperties.getProperty("topology.kafka.json.file.spout4");
            try {
                KafkaProducerUtil kafkaProd1 = new KafkaProducerUtil();
                KafkaProducerUtil kafkaProd2 = new KafkaProducerUtil();
                KafkaProducerUtil kafkaProd3 = new KafkaProducerUtil();
                KafkaProducerUtil kafkaProd4 = new KafkaProducerUtil();
                kafkaProd1.readJsonAndSendMsg(jsonFileSpout1, topicname1);
                kafkaProd2.readJsonAndSendMsg(jsonFileSpout2, topicname2);
                kafkaProd3.readJsonAndSendMsg(jsonFileSpout3, topicname3);
                kafkaProd4.readJsonAndSendMsg(jsonFileSpout4, topicname4);
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }
        // create rtDeep Topology
        rtDeepTopology rtdeepTopology = new rtDeepTopology(generalProperties);
        // get general topology configuration parameters
        int topologyNumWorkers =
        Integer.valueOf(generalProperties.getProperty("topology.num.workers"));
        boolean debugEnabled =
        Boolean.valueOf(generalProperties.getProperty("topology.enable.debug"));
        String topologyName = generalProperties.getProperty("topology.name");
        int msgTimeOut =
        Integer.valueOf(generalProperties.getProperty("topology.set.message.timeout.secs"));
        int topologyTimeOut =
        Integer.valueOf(generalProperties.getProperty("topology.time.out"));
        //  set up configuration object with the properties retrieved
        Config conf = new Config();
        conf.registerSerialization(TupleApi.class);
        conf.setDebug(debugEnabled);
        conf.setNumWorkers(topologyNumWorkers);
        conf.setMessageTimeoutSecs(msgTimeOut);
        try {
            topologyTest.log.info("Running Local topology...");
            LocalCluster cluster = new LocalCluster();
            // Submit topology to local cluster
            cluster.submitTopology(topologyName, conf,
            rtdeepTopology.create().createTopology());
            Thread.sleep(topologyTimeOut);
            cluster.shutdown();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
#######################################
#
# Rt Deep Topology properties v1.0
# Author: Roberto Robledo McClymont
#
#######################################
# GENERAL PROPERTIES
#######################################
#topology name
topology.name=rt-deep-topology
# number of workers
topology.num.workers=4
topology.enable.debug=false
topology.time.out=300000
topology.set.message.timeout.secs=1200000
# enable or disable if you want to fill in kafka topic before storm execution
topology.fill.kafka.topics=true
topology.kafka.json.file.spout1=src/test/resources/inputJSONKafkaTopic/testSpout1.json
topology.kafka.json.file.spout2=src/test/resources/inputJSONKafkaTopic/testSpout2.json
topology.kafka.json.file.spout3=src/test/resources/inputJSONKafkaTopic/testSpout3.json
topology.kafka.json.file.spout4=src/test/resources/inputJSONKafkaTopic/testSpout4.json
#######################################
# INPUTS KAFKA = SPOUTS
#######################################
#Define input topic kafka channel names
kafkaspout1.topicname=observatory1
kafkaspout1.id=kafka-spout-ob1
kafkaspout2.topicname=observatory2
kafkaspout2.id=kafka-spout-ob2
kafkaspout3.topicname=observatory3
kafkaspout3.id=kafka-spout-ob3
kafkaspout4.topicname=observatory4
kafkaspout4.id=kafka-spout-ob4
zookeeper.host=localhost
zookeeper.port=2181
#######################################
# FILTER INPUT KAFKA SPOUT
#######################################
## define configuration files and ids for filter kafka input spouts
bolt1.filter.spout1.config.path=src/test/resources/filterInput/boltFilterSpout1Config.properti
es
bolt.filter.spout1.id=bolt-filter-spout1
bolt2.filter.spout2.config.path=src/test/resources/filterInput/boltFilterSpout2Config.properti
es
bolt.filter.spout2.id=bolt-filter-spout2
bolt3.filter.spout3.config.path=src/test/resources/filterInput/boltFilterSpout3Config.properti
es
bolt.filter.spout3.id=bolt-filter-spout3
bolt4.filter.spout4.config.path=src/test/resources/filterInput/boltFilterSpout4Config.properti
es
bolt.filter.spout4.id=bolt-filter-spout4
#######################################
# JOIN INPUTS BOLT
#######################################
## define configuration from joinInputs
joiner.inputs.id=joiner-kafkaSpout
joiner.inputs.key.field=id
#######################################
# FILTER TRAINING - TEST MODE BOLT
#######################################
## define configuration files and ids for filter training test bolt
bolt.filter.training.and.test.config.path=src/test/resources/filterTrainTest/boltFilterTranini
ngTestConfig.properties
bolt.filter.training.and.test.id=bolt-filter-training-test
#######################################
# CLASSIFIERS BOLTS
#######################################
## define configuration files and ids for classifiers bolts
```

```
# 0 : standard condition
# 1 : tsunami
classifier.class.label.list=0,1
bolt1.classifier.config.path=src/test/resources/classifier/bolt1ClassifierConfig.properties
bolt1.classifier.id=bolt1-mcnn-classifier
bolt2.classifier.config.path=src/test/resources/classifier/bolt2ClassifierConfig.properties
bolt2.classifier.id=bolt2-mcnn-classifier
bolt3.classifier.config.path=src/test/resources/classifier/bolt3ClassifierConfig.properties
bolt3.classifier.id=bolt3-mcnn-classifier
bolt4.classifier.config.path=src/test/resources/classifier/bolt4ClassifierConfig.properties
bolt4.classifier.id=bolt4-mcnn-classifier
######################################
# JOIN CLASSIFIER BOLT
######################################
## define configuration for joinClassifiers
joiner.classifier.id=joiner-classifier
joiner.classifier.key.field=id
######################################
# COMPUTE MAJOR VOTE BOLT
######################################
## define configuration files for compute major vote
bolt.compute.major.vote.config.path=src/test/resources/computeMajorVote/computeMajorVoteConfig
.properties
bolt.compute.major.vote.id=bolt-compute-major-vote
######################################
# OUTPUT KAFKA
######################################
#define output kafka channel topic properties
kafka.output.id=kafka-output-result
kafka.ouput.topicname=result-label-predict
kafka.output.config.path=src/test/resources/kafka/kafkaProducer.properties
```

```
bolt.input.fields=id,label
bolt.output.fields=id,label
bolt.input.fields=id,label
bolt.output.fields=id,label
bolt.input.fields=id,label
bolt.output.fields=id,label
bolt.input.fields=id,label
bolt.output.fields=id,label
bolt.input.fields=id,label
bolt.output.fields=id,winner
bolt.input.fields=id,item1,item2,item3,item4
bolt.output.fields=id,tuple
bolt.input.fields=id,item1,item2
bolt.output.fields=id,tuple
bolt.input.fields=id,item1
bolt.output.fields=id,tuple
bolt.input.fields=id,item1,item2,item3,item4,item5
bolt.output.fields=id,tuple
bolt.input.fields=id,tuple
bolt.output.fields=tuple,processId
```

```
#kafka producer properties
bootstrap.servers=localhost:9092
acks=1
key.serializer=org.apache.kafka.common.serialization.StringSerializer
value.serializer=org.apache.kafka.common.serialization.StringSerializer
```

```
#spout 1 and spout 2 follow a normal distribution PDF
random.normal.mean=0.0010,0.0020
random.normal.std=0.001,0.001
#spout 3 and spout 4 follow a uniform distribution PDF
random.uniform.lower=0.010,0.002
random.uniform.upper=0.020,0.004
class.label=0
```

```
#spout 1 and spout 2 follow a normal distribution PDF
random.normal.mean=0.0010,0.0020
random.normal.std=0.001,0.001
#spout 3 and spout 4 follow a uniform distribution PDF
random.uniform.lower=0.001,0.008
random.uniform.upper=0.002,0.010
class.label=1
```

```
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-topics.sh --list --zookeeper localhost:2181
```

```
# produce msg
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-console-consumer.sh --topic observatory1
--from-beginning --zookeeper localhost:2181
```

```
# produce msg
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-console-consumer.sh --topic observatory2
--from-beginning --zookeeper localhost:2181
```

```
# produce msg
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-console-consumer.sh --topic observatory3
--from-beginning --zookeeper localhost:2181
```

```
# produce msg
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-console-consumer.sh --topic observatory4
--from-beginning --zookeeper localhost:2181
```

```
# produce msg
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-console-consumer.sh --topic result-label-predict
--from-beginning --zookeeper localhost:2181
```

```
# create topic
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-topics.sh --create --topic observatory1
--replication-factor 1 --partitions 1 --zookeeper localhost:2181
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-topics.sh --create --topic observatory2
--replication-factor 1 --partitions 1 --zookeeper localhost:2181
```

```
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic
observatory1
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic
observatory2
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic
observatory3
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic
observatory4
```

```
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic
result-label-predict
```

```
ZK_IP=$(sudo docker inspect --format '{{ .NetworkSettings.IPAddress }}' zookeeper)
KAFKA_IP=$(sudo docker inspect --format '{{ .NetworkSettings.IPAddress }}' kafka)
# produce msg
/opt/Kafka/kafka_2.11-1.0.1/bin/kafka-console-producer.sh --topic test --broker-list
$KAFKA_IP:9092
```

```
sudo kill -9 $(pgrep -f zookeeper)
# run zookeeper server
sudo /opt/Kafka/kafka_2.11-1.0.1/bin/zookeeper-server-start.sh -daemon
/opt/Kafka/kafka_2.11-1.0.1/config/zookeeper.properties
# run kafka server
sudo /opt/Kafka/kafka_2.11-1.0.1/bin/kafka-server-start.sh -daemon
/opt/Kafka/kafka_2.11-1.0.1/config/server.properties
# STORM user interface
#sudo /opt/apacheStorm/apache-storm-1.2.1/bin/storm nimbus &
#sudo /opt/apacheStorm/apache-storm-1.2.1/bin/storm supervisor &
#sudo /opt/apacheStorm/apache-storm-1.2.1/bin/storm ui &
```

```
# MDS Plot Synthetic
# 14/04/18
# Roberto Robledo McClymont

import numpy as np
import csv

from sklearn.preprocessing import scale
from sklearn.manifold      import MDS

import collections
import matplotlib.pyplot as plt
import math

# Read the data file, return a np.array with attribs and a list of class labels
#and the string list of name of each column
def readData(filename):
    with open(filename, newline='') as csvfile:
        myreader = csv.reader(csvfile)
        attribs = []
        # last column class label
        indexStatus =12
        classes = []
        for row in myreader:
            #delete indexStatus column
            classes.append(row[indexStatus])
            row.pop(indexStatus)
            ## skip status that corresponds to the class
            ## skip also name identifier index 0
            attribs.append([float(e) for e in row[:]])
    return np.array(attribs), classes

 # Apply MDS and plot
def mdsplot(mydata, classes,filename):
    myMDS      = MDS(n_components = 12)
    projection = myMDS.fit_transform(mydata)
    # Plot
    fig = plt.figure()
    fig.suptitle('MDS projection')
    plt.scatter(projection[:,0],projection[:,1],marker='o',c=classes)
    plt.savefig(filename)

# Read and prepare the data and run the different activities of the exercise
if __name__ == '__main__':

    print ("Reading Synthetic's file ...")
    # -------------- Data loading and preprocessing
    mydata, myclasses= readData('dataSynthetic.csv')
    totalObservations = mydata.shape[0]
    print ("Number of observations : " + str(totalObservations))
    print ("Number of attributes/variables : " + str(mydata.shape[1]))
    # Scale the data (it is not mandatory as PCA already does it by default)
    mydata2 = scale(mydata)
    mdsplot(mydata2,myclasses,'MDS0.png')
```

```
echo "Total of predicted tuples:"
cat log.log | grep -e MajorVote.=. | wc -l
echo "Number of MajorVotes of class label one:"
cat log.log | grep -e MajorVote.=.1 | wc -l
echo "Number of MajorVotes of class label zero:"
cat log.log | grep -e MajorVote.=.0 | wc -l
```