Real Time Predictive Big Data Analysis With Streaming ML Support for WSO2 ML (Machine Learner) and CEP (Complex Event Processor)

Problem

When it comes to big data we are serving for vital four big Vs [1]: Volume (Scale of Data), Variety (Different Forms of Data), Velocity (Analysis of Streaming Data), Veracity (Uncertainty of Data). Today it is paramount important to handle large volume of data with high velocity which becomes more important paradigms than the variety and veracity which are handled my high end machine learning algorithms. When we think of the big data, we probably think of massive database of millions or even billions of files, which will no longer be a complementary solution as the volume of the data is increasing rapidly. With concept of big data usual standalone data analysis may not going to work, because of the latency and storage bottleneck that cannot be stand with the future volume and velocity. Though there are prevails modern techniques such as batch processing which is an efficient way to generate insights when working with a high volume of data. On the other hand MapReduce paradigms, which divides massive datasets across multiple clusters of commodity hardware, is the classic example of efficient batch processing with machine learning [2].

Recently with the development of IoT (Internet of Things), the story is becoming different where the velocity and volume is becoming vital as long as the most important concern is to extracting near real-time insights from massive amount of data much faster than earlier. Therefore providing flexible and scalable data analytics for streaming by using various techniques such as mini-batch processing is becoming more importance to the technology. As long as distributed computing is advancing, mini-batch is very efficient to train models and retain them.Initial problem is to extend the WSO2 current features into streaming data analysis that can retrain and persist machine learning models by using techniques such as mini-batch processing as well as high end machine learning algorithms such as Stochastic Gradient Descent (SGD) Optimizations. Because though they support standalone predictions with ml models with their SIddhi: Complex Event Processor, currently WSO2 ML does not provide predictions with retaining machine learning models at real time. One major fact in this consideration is persist the information and insight of the previous machine learning models while retain them with new mini-batches. This way we can avoid storing all the data across time, by persisting ML models, not the previous data. In that case retaining ML model contains information and insight of the previous data. This is much important when when we comes to

massive online predictive data analysis, where we predict while retraining with most recent data.idea of incremental learning with streaming data focuses on two objectives:

1. Identifying patterns in the recent history.
2. Updating the patterns with incoming data without catastrophic forgetting

To achieve this we could use several techniques combined with their native algorithms to train and retrain the ML models. We had two options.

1. Incremental algorithms - there are machine learning algorithms which can be modified to support incremental learning. eg. mini-batch k-means, stochastic gradient descent based linear models, SVM, etc.
2. Periodic re-training - machine learning models are trained with buffered data periodically.

The most efficient and streaming aware techniques is the Incremental Learning Algorithms which can be used to reduce the size of the data storage. In this incremental learning model which perceive the insight of the all past data that arrives to the training occasions. In this kind of streaming learning approach we need flexibility, adaptability and scalability to achieve the explicit objective of our predictive big data analytics. When it comes to predictive analysis with streaming support we have to keenly consider on aspects such as data horizon (how quickly a most recent data point becomes a part of the model) and data obsolescence (how long does it take a past data point to become irrelevant to the model). These are aspects are vital in stream mining and because of that it is paramount important to keep them adjustable to users.

And in the case of WSO2, this solution should be used as API/Extension to work with CEP siddhi streams as well as other entrance interfaces from various sources like different file systems. CEP siddhi is apache storm based stream processing framework, which can be easily used manipulate streams. In that case this solution should be worked as extension to siddhi for massive predictions and analysis of data real time by taking decision based on existing model and feeding data points/events. Thus we do not need to save data all time in a massive data bases and insert them at once and forget previously trained ML (Machine Learning) models while new data is coming which may be the current scenario.

Solution

The solution is to make better use of prevailing incremental machine learning algorithms such as SGD (Stochastic Gradient Descent) , Mini Batch algorithms and kernel based algorithms to facilitate undisputed re-train of ML models to perceive the insight of data even as streams with fast learning and predicting cycles. Decision was made upon framework which can be used to provide mini-batch learning which can be extended for streaming as well. According to need and flexibility for online prediction,to come up with a good solution and incremental learning algorithms i have to understand what is the difference between mini-batch processing and streaming algorithms. These are two concepts.

One solution is purely mini-batch processing which can be extended for streaming and time sensitive analysis by reducing batch size or time-window where volume of the data is

handles much better way. On the other hand other solution is purely streaming aware which can be used for micro-batches as well.

Batch processing is an efficient way to generate insights when working with a high volume of data [2], which divides massive datasets across multiple clusters. .**Batch processing** is ideal for generating this kind of **strategic insight** when there's no particular advantage or need to process this data in real time. To perform real time as stream data we have to reduce the batch size or time-window size that the model will be updated.

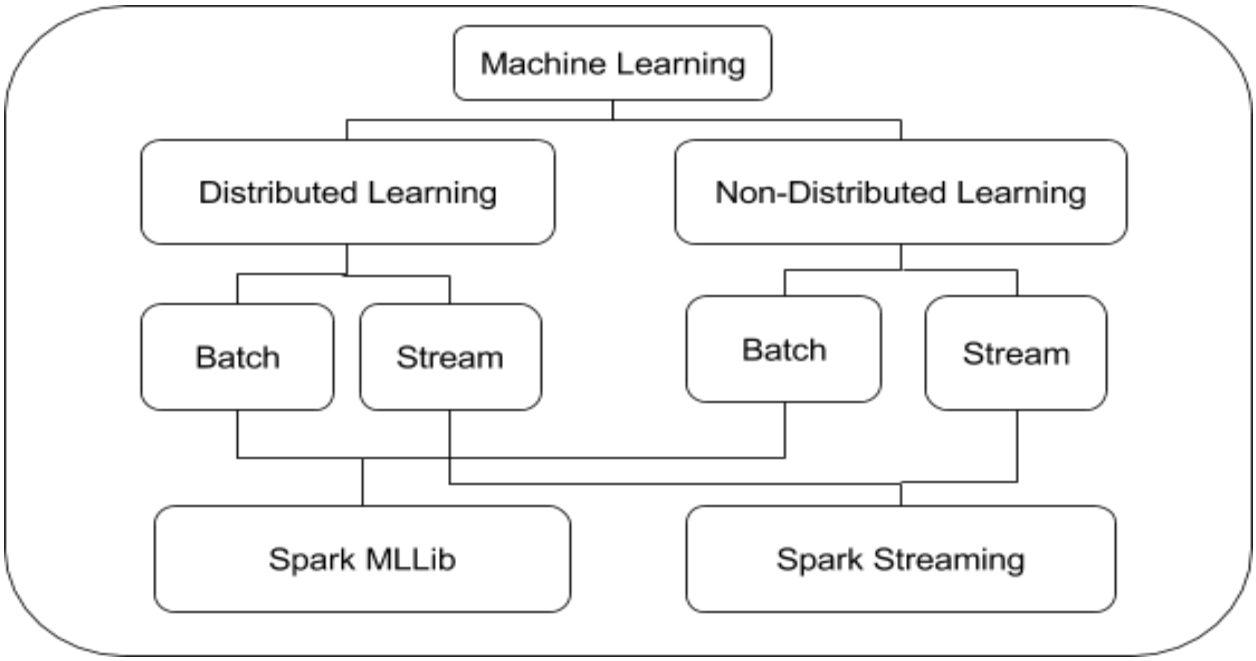
But When your most important consideration is extracting near real-time insights from massive amounts of data, you need **streaming data.** Streaming data is meant to process and analyze large and constantly moving volumes of data. There are two main approaches for streaming data. [2] The first approach is called **native stream processing** (or tuple-at-a-time processing). In this approach, every event is processed as it comes in, one after the other, resulting in the lowest-possible latency. Unfortunately, processing every incoming event is also computationally expensive. **Micro-batch processing** makes the opposite tradeoff, dividing incoming events up into batches either by arrival time or until a batch has reached a certain size. This reduces the computational cost of processing but can also introduce more latency.

This is why we have two solutions though we promised for single solution in the GSOC proposal. These two solutions are exactly focusing on following aspects. One solution is Apache Spark based incremental learning with mini-batch processing which can be used for streaming prediction analysis. On the other hand Apache Samoa based massive predictive analysis as stream data which can be used with micro-batch processing as well. Both have apache storm for distributed stream processing framework. In advance Apache samoa can have both native stream processing and Micro-batch processing while it can be run on advanced stream processing platforms such as Samza. Both solutions is important when it comes to online predictive analysis with massive data.

Initial Solution Based on the Apache Spark

Spark core can be used both non distributed and distributed machine learning which provide better simplicity, scalability, compatibility and streaming awareness. Apache spark has two main machine learning methods on top of their core: Spark Mllib and Spark Streaming. If we take Apache spark architecture.

With the emergence of distributed computing like storm stream processing platforms the mini-batch processing is vital concerning volume of the big data, But add some latency on updating Machine Learning (ML) models. Paradigms such as MapReduce is used to create mini-batches in an efficient way avoiding large databases which divides massive datasets across multiple clusters of commodity hardware. Where different frameworks have different algorithms for MapReduce. But in advanced Apache Spark have much faster and with reduced latency for MapReduce compared to Hadoop framework. In batch-processing the time taken is depending on the size of the batch/job. [2] But if our most important consideration is extracting near real-time insight from massive amount of data, we can use streaming processing. But in the same time by reducing batch-size or time-window size we can use mini-batch processing for streaming as well.



As an initial idea of WSO2 ML team, the solution was to implement incremental algorithms which retrain machine learning models based on the mini-batch algorithms. Therefore the initial deliverables was to implement Streaming aware Linear Regression with Stochastic Gradient Descent (SGD) algorithms and Mini-batch KMeans clustering with the Spark support. Mini-Batch algorithms also can be used in two different ways. One method is to process as original batches without any overlapping between mini-batches. Another method is to introduce moving window that can be move over sample data points and train the model. In both cases the model is retrained and updated according to batch-size. Other technique that we can use to update or retrain the ML model is introducing time-window rather than batch-window or moving-batch-window. Initially our target was to implement API for Linear regression with SGD optimization and Mini-Batch Streaming Clustering which can be used for streaming application as well .To make this more streaming aware we planned to introduce the adoptable parameters such as follows. If we consider Streaming Linear regression with SGD optimization,

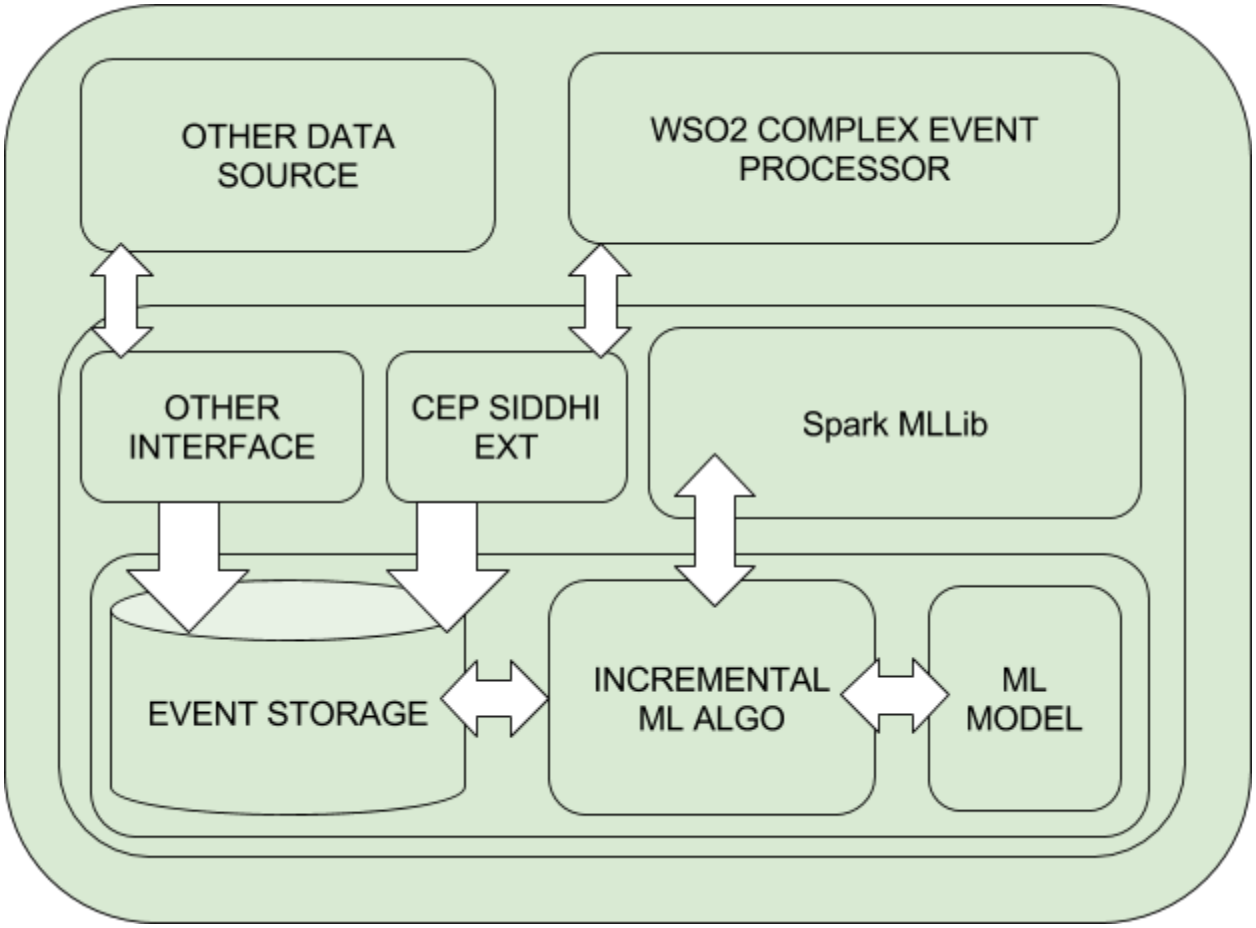
Learn Type: Learning method (batch-window, moving-batch-window, time-window)
Window-Shift: Apply only when the moving batch window is applied
Mini-Batch - Size: How often the ML model get retrained or updated
Number of iteration: Number of iteration runs on the SGD algorithms
Step-Size: related to Step Size of the SGD optimization algorithm.
Mini-Batch Fraction: Fraction of the mini-Batch to process at every iteration in SGD algorithm.
Confidence Interval : This is for future use if there will be any requirement
Variable-List : feature Vector of the Data point

And for the Mini-Batch K Means Clustering we can parameterize it as the same as above like,

Learn Type: Learning method (batch-window, moving-batch-window, time-window)
Window-Shift: Apply only when the moving batch window is applied
Mini-Batch - Size: How often the ML model get retrained or updated
Number of iteration: Number of iteration runs on the SGD algorithms
Number of Clusters: Number of Clusters need for us
Alpha : Decay Factor which can be used for Data obsolescence
Confidence Interval : This is for future use if there will be any requirement
Variable-List : feature Vector of the Data

So we have decided to use apache Spark MLLib algorithms for our initial implementation which is very realistic for massive online analysis in terms of volume because Spak has the fastest MapReduce algorithm when compared to hadoop. Spark also provide spark streaming which can be used with these mini-batch algorithms such as SGD with distributed support for processing.Spark Streaming [3] is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data stream. But WSO2 already has complex event processor which can be used to feed data as streams to implemented API. Therefore in our first API we focused on developing core which can retrain models with incremental algorithms where the data/events can be fed as many forms such as streams, files etc. Therefore additionally i had to integrate my API with the WSO2 CEP siddhi extension to feed data for train and prediction models for cep online predictions.

Architecture



Core Architecture is as Incremental ML core with WSO2 Siddhi extension for predictive analysis. As long as Spark had mini-batch processing for data it has to make mini-batch of data to train ML model inside Incremental ML algorithms. Based on the spark with the same architecture as above i have implemented two Streaming aware ML modules: Streaming Linear Regression and Streaming K Means Clustering.

Streaming Linear Regression with SGD

Streaming Linear Regression based on the SGD (Stochastic Gradient Descent) Optimization is implemented in the API as StreamingLinearRegression.java class with other relevant classes such as StreamingLinearRegressionModel.java .

StreamingLinearRegression.java

Constructor:

StreamingLinearRegression(int learType, int windowShift, int paramCount, int batchSize, double ci, int numIteration, double stepSize, double miniBatchFraction)

Function	Input List	Output Type	Description
regress()	Double[]event	Object[]output	Regress with model retrain as mini-batches. This has three counterparts inside. Regress as Batches, Moving Window & Time Window.
buildModel()	List<String>events	Object[]output	Train Model Using String List of data points/events as mini batches.
getRDD()	List<String>events	JavaRDD<LabeledPoint>	Convert Set of String Events into Spark RDD (Resilient Distributed Dataset)
getMSE()	JavaRDD<LabeledPoint>parsedData, LinearRegressionModel buildModel	Double mse	Calculate MSE (Mean Square Error) if the data set with the build regression model
trainData()	JavaRDD<LabeledPoint>parsedData, int stepSize, int numIteration, int miniBatchFraction	LinearRegressionModel model	Train a new Model forgetting past model with fresh set of data/events
retrainModel()	JavaRDD<LabeledPoint>parsedData, int stepSize, int numIteration, int miniBatchFraction	LinearRegressionModel model	Train a new Model forgetting past model with fresh set of data/events

In this spark based implementation the preparing min-batch is done inside the module. Data Types for the relevant parameters is as follows,

Learn Type: Learning method (batch-window, moving-batch-window, time-window)- Integer
Window-Shift: Apply only when the moving batch window is applied - Integer
Mini-Batch - Size: How often the ML model get retrained or updated - Integer
Number of iteration: Number of iteration runs on the SGD algorithms - Integer
Step-Size: related to Step Size of the SGD optimization algorithm - Double
Mini-Batch Fraction: Fraction of the mini-Batch to process at every iteration - Double
Confidence Interval : This is for future use if there will be any requirement - Double
Variable-List : feature Vector of the Data point

Streaming Linear Regression WSO2 CEP Siddhi Extension for Event Streams

In this API we support WSO2 CEP (Complex Event Processor) Siddhi Extension as one interface to feed data into our Streaming ML modules. Example Siddhi Query to invoke this module is as follows.

Query Structure

If you are using standalone extension toy can use,

streamingml:streamlinreg([learn-type], [window-shift], [mini-batch-size],[number-of-iteration], [step-size], [mini-batch-fraction], [ci], {Variable List})

Or if you are using carbon-ml extension,

ml:streamlinreg([learn-type], [window-shift], [mini-batch-size],[number-of-iteration], [step-size], [mini-batch-fraction], [ci], {Variable List})

Learn-Type

0 - Mini-Batch Processing

1- Mini-Batch Processing with Moving Window with Moving Shift Variable

2 - Time - Window Based Mini-Batch Processing (For Future Implementation)

Mini-Batch-Fraction

0 - 1 : percentage between 0-1

Ci (confidence-interval)

Confidence Interval is not using in algorithms for now. Keep it for future as optional field for algorithms.

Example Query

```
@Import('ccppInputStream:1.0.0')
define stream ccppInputStream (PE double, ATV double, V double, AP double, RH double);

@Export('ccppOutputStream:1.0.0')
```

```
define stream ccppOutputStream (stderr double);

from ccppInputStream#streamingml:streamlinreg(1, 2, 4,10, 0.00000001, 1.0, 0.95, PE, ATV, V,
AP, RH)
select stderr
insert into ccppOutputStream;
```

Hint: Use The CCPP data set from UCI repository] to directly use above queries.

Input CEP Event Streams

Input Stream should contain the variable in a way that first variable should be the dependent variable and other variables should be independent variables.

```
@Import('ccppInputStream:1.0.0')
define stream ccppInputStream (PE double, ATV double, V double, AP double, RH double);
```

Here PE is the dependent variable where ATV, V, AP, RH are independent variables for the regression analysis.

Output CEP Event Streams

Output when there is a new model or updated model will be sent with a mse error and model parameters. As an example if we have p+1 variables in the variable list like
[var0, var1…., varp]

Then the output stream from extension side is like
[var0, var1, ….., varp, stderr, beta0, beta1…., betap];
Beta0 is the intercept of the trained/retrained regression model

Apache Spark based implementation is more towards volume of the data, which can simplify massive databases into pieces which can be processes distributed parallel at run time.

Streaming K Means Clustering with Mini-Batch Processing

Streaming KMeans Clustering is also a part of the API which can be accessed via CEP extension. Since we were moving to SAMOA based streaming predictive analysis basic extension and clustering functions are completed.

Constructor:
StreamingKMeansClustering(int learnType, int windowShift, int numAttributes, int batchSize, double ci, int numClusters, int numIterations, double alpha)

Functions:

Function	Input	Output	Description
cluster()	Double[]eventData	Object[]output	Clustering with data point and when the model is updated put into the output array. Here also cluster as mini-batches, mowing-window,time-window use
buildModel()	List<String>eventsMem	Object[]output	Build Clustering Model with the mini-batch in the form of List<String> and build a StreamingKMeansClusteringModel
getRDD()	List<String>events	JavaRDD<Vector>parsedData	Covert List of String data point/ events in the form of comma separated version into vectors of data set in RDD (Resilient Distributed Dataset)
getWSSSE()	JavaRDD<Vector> parsedData, KMeansModel model	Double wssse	Calculate WSSSE with the mini-batch and trained ML model
trainData()	JavaRDD<Vector> points	KMeansModel model	Build new ML model with the data points in the mini-batch
retrainModel()	StreamingKmeansClusteringModel prevModel, STreamingKMeansClusteringModel newModel , int numClusters	StreamingKMeansClustering model (Updated/Retrained Model)	Retrained the KMeans Model is based on the mini-batch clustering. First we need to build the fresh model with the dataset using trainData(). Then we have to combine the new model with the previous model.
getClusterWeights()	JavaRDD<Vector> events, KMeansModel model, int numClusters	Vector weights	Calculate the weights of the each cluster and put it into the vector of size=numClusters.

The parameter definitions and the type for each parameter that is even used in the cep extension is as follows.

Learn Type: Learning method (batch-window, moving-batch-window, time-window) - Integer

Window-Shift: Apply only when the moving batch window is applied - Integer
Mini-Batch - Size: How often the ML model get retrained or updated - Integer
Number of iteration: Number of iteration runs on the SGD algorithms - Iteration
Number of Clusters: Number of Clusters need for us - Integer
Alpha : Decay Factor which can be used for Data obsolescence - Integer
Confidence Interval : This is for future use if there will be any requirement - Double
Variable-List : feature Vector of the Data - Attributes

Streaming KMeans Clustering WSO2 CEP Siddhi Extension

Query Structure

If you are using standalone streaming xml extension,
streamingml:streamclustering([learn-type], [window-shift], [batch-size], [number-iterations], [number-clusters], [alpha], [confidence-interval], PE, ATV, V, AP, RH)

If you are using carbon-ml extension use,
ml:streamclustering([learn-type], [window-shift], [batch-size], [number-iterations], [number-clusters], [alpha], [confidence-interval], PE, ATV, V, AP, RH)

Learn-Type

0 - Mini-Batch Processing

1- Mini-Batch Processing with Moving Window with Moving Shift Variable

2 - Time - Window Based Mini-Batch Processing (For Future Implementation)

Window-shift

Only apply when the learn type is 1 - mini-batch processing with the moving window

Number of Clusters

Number of KMeans Clusters need where the output streams contain that number of cluster centers for predictions and analysis.

Alpha

Alpha is used as a decay factor when combining the new model with the previous model when retrain the ML model. Alpha is native parameter of the mini-batch clustering. In this retrain scheme the new cluster centers will be defines as follows.

$$c(n+1) = (c(t) * n(t)) * alpha + x(t) * m(t))/(n(t) * alpha + m(t))$$

$$n(t+1) = n(t) + m(t)$$

[3] Where $c_{(t)}$ is the previous center for the cluster, $n_{(t)}$ is the number of points assigned to the cluster thus far, x_t is the new cluster center from the current batch, and $m_{(t)}$ is the number of points added to the cluster in the current batch. The decay factor α can be used to ignore the past: with α=1 all data will be used from the beginning; with α=0 only the most recent data will be used.

Ci (confidence-interval)
Confidence Interval is not using in algorithms for now. Keep it for future as optional field for algorithms.

Query Example

```
@Import('ccppInputStream:1.0.0')
define stream ccppInputStream (PE double, ATV double, V double, AP double, RH double);

@Export('ccppOutputStream:1.0.0')
define stream ccppOutputStream (PE double, ATV double, V double, AP double, RH double, stderr double, center0 string, center1 string);

from ccppInputStream#streaming:streamclustering(0, 0, 1000, 10, 2, 1, 0.95, PE, ATV, V, AP, RH)
select *
insert into ccppOutputStream;
```

Hint: This query is used for the CCPP dataset from UCI repository.

Input CEP Event Streams

Input stream to the streaming KMeans Clustering extension is consist of the all features of the sample. Since this is unsupervised learning there are no dependent variables.

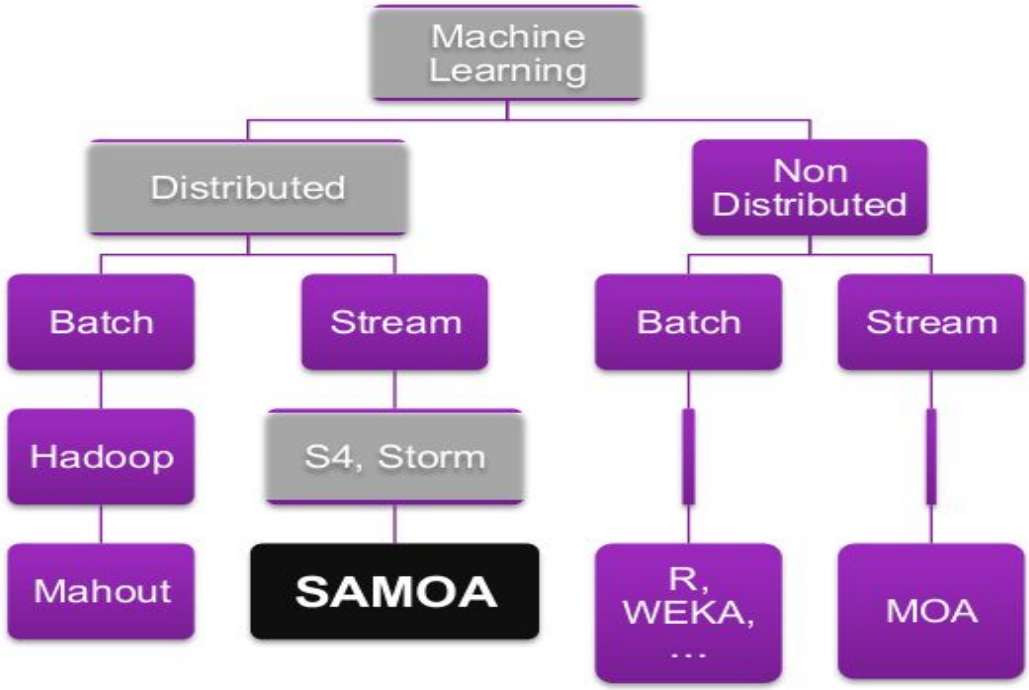
```
@Import('ccppInputStream:1.0.0')
define stream ccppInputStream (PE double, ATV double, V double, AP double, RH double);
```

Output CEP Event Streams

Output when there is a new model or updated model will be sent with a WSSSE and cluster centers. As an example if we have k clusters, k+1 the output stream is looks like follow,

[var0, var1, ….., varp, stderr, cluster0, cluster11…., clusterk];

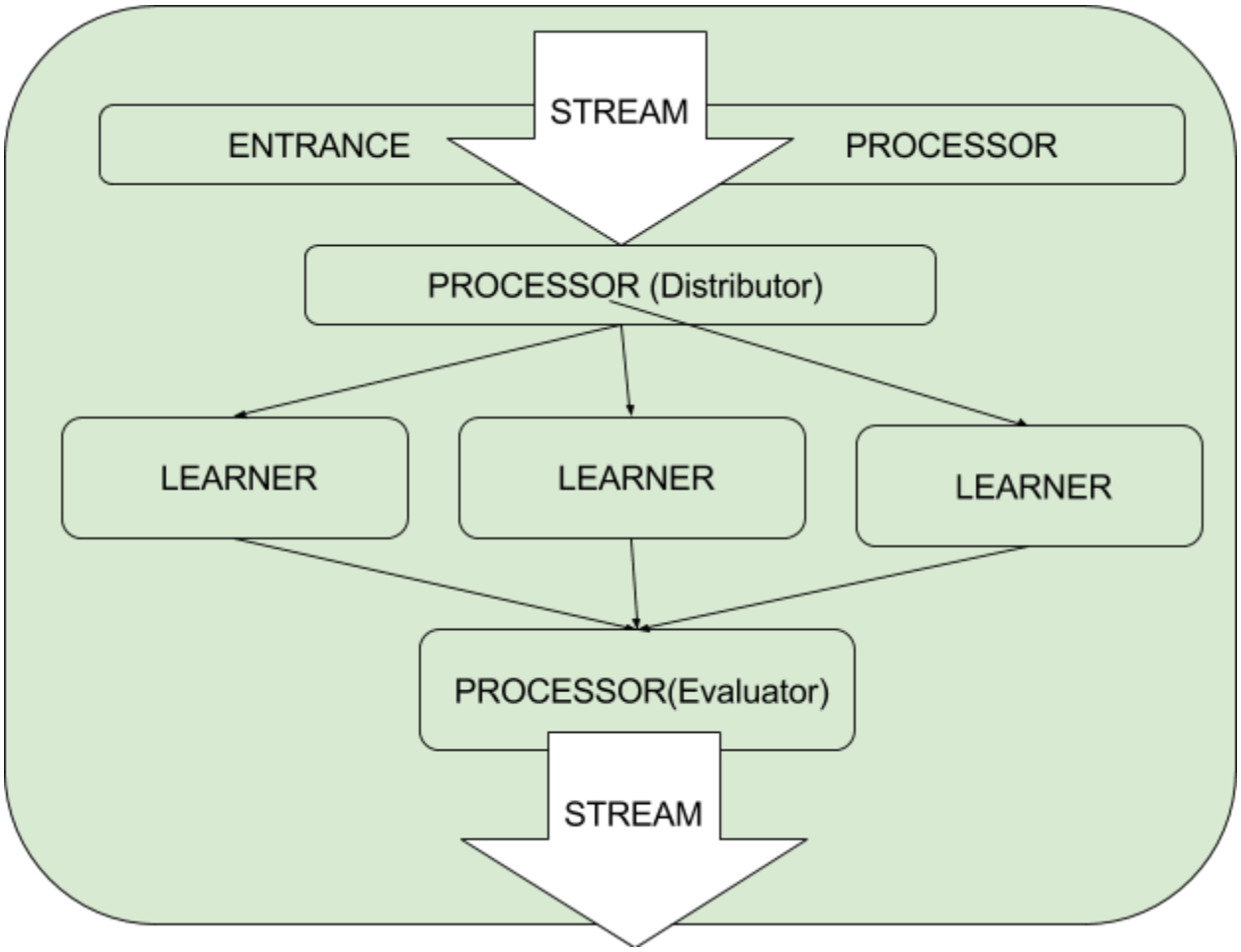
Streaming Solution Based on the Apache SAMOA



Overview

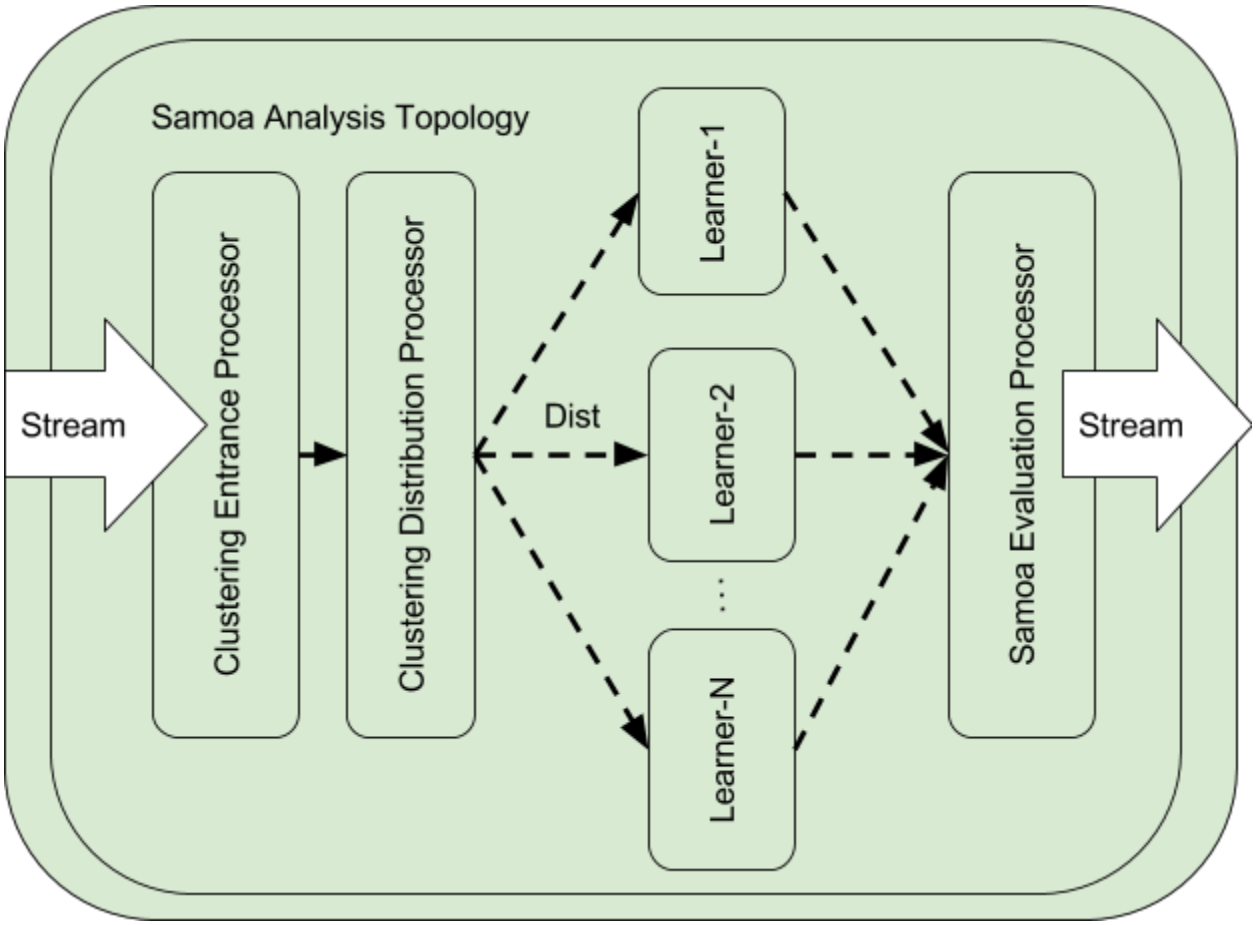
SAMOA is highly scalable platform for distributed Streaming predictive analysis. Its architecture is based on streaming analysis which can be used for micro-batches as well. SAMOA is scalable because of its simple architecture that can be built upon basic SAMOA building blocks. [5] Samoa can be run on all main three streaming processing framework, storm, samza and p4. As the name implies SAMOA - Scalable Advanced Massive Online Analysis, it was developed to handle both velocity as well as volume of the big data for predictions.

Samoa build distributed topologies by using basic building blocks and to handle stream data for prediction. To get a basic understanding of SAMOA topologies please refer to [6]. Streaming ML topologies are built upon the basic building blocks of Processors, Entrance Processors, Streams, Learners, Task, Topology Builder, Content Event etc.



SAMOA is like a Complex Event Processor consist of predictive ML topologies inside.Streams Consist of Content Events and Processor Consist of Processing Elements. For more information please refer [6].

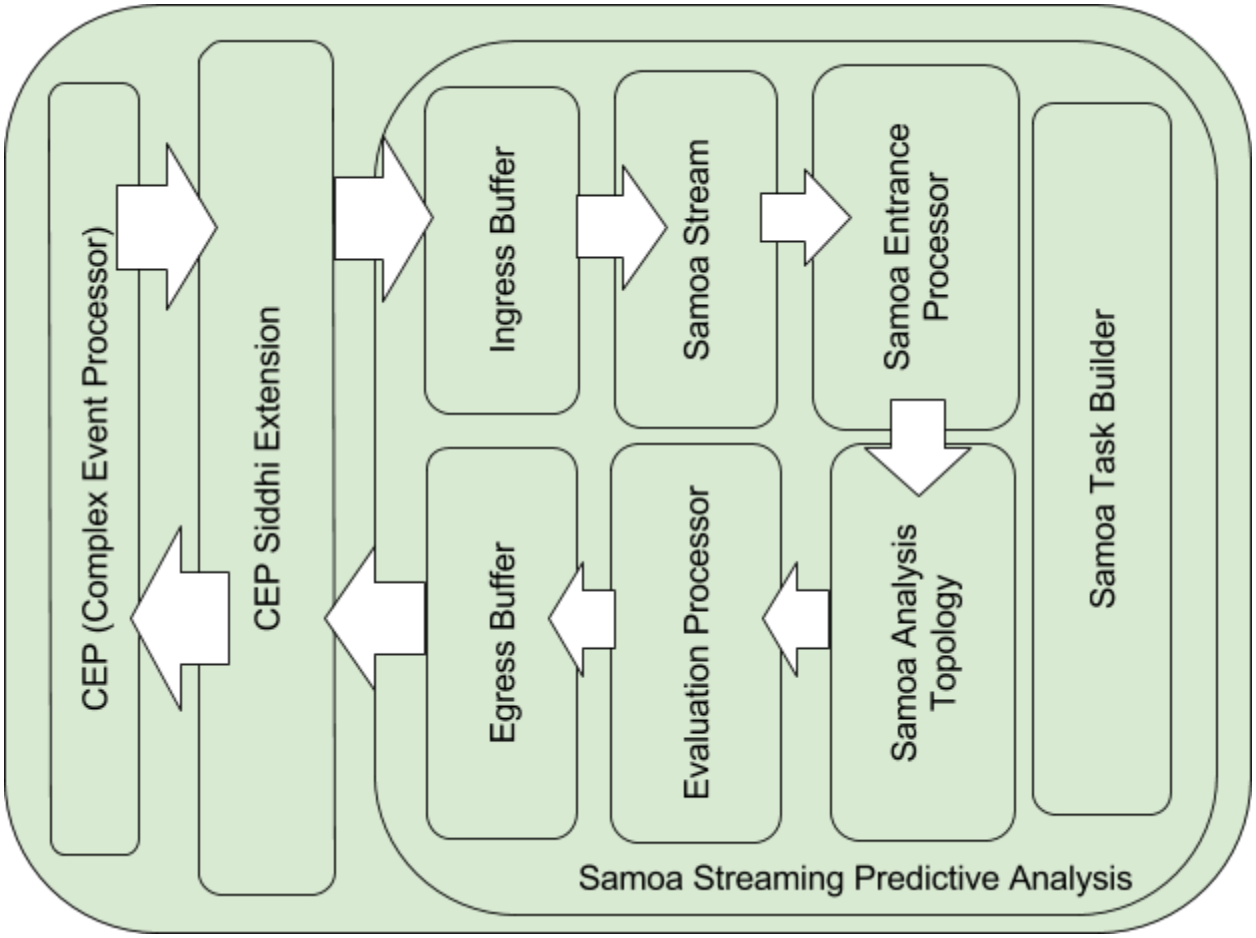
Architecture



Integration of SAMOA (Scalable and Massive Online Analysis) with WSO2 CEP is a challenging task. To do that i had to go through both CEP architecture and SAMOA architecture. In Samoa by using their basic building blocks we can create new predictive analysis ML topologies to train ML models learn from data by extracting insight of near massive data. To integrate SAMOA with the new framework such as WSO2 CEP, we should have a good knowledge to handle samoa blocks. By this way we can feed data coming from the CEP to SAMOA ML topologies built for our purposes. To make use of the SAMOA for custom integration please look at samoa here. For CEP and our API integration SAMOA, customized SAMOA modules are used which we will discussed in the Implementation. In the SAMOA integration its is paramount important to decide the architecture for core to handle data as streams for real time learning. You can find a custom SMOA Task which can be build with basic SAMOA component in the following.

Samoa Learning Topologies

Overall Architecture of the SAMOA based implementation is looks like below when SAMOA is integrated with CEP extension to feed data streams. Because of the simple and distributed nature of the SAMOA, it is very easy to build complex and massive streaming analysis topologies with SAMOA building blocks. As a initial steps we have build a SAMOA streaming clustering topology as shown in the above figure to analyses stream data online. More details of the implementation can be found in Implementation.



Both implementations are JAVA based implementations. Samoa is using instances and InstanceStreams not Events and EventStreams like in CEP. Therefore stream data is buffered and convert from CEP events to SAMOA instances for analysis. CEP has their native ExecutionPlans to invoke predictive analysis which is used to build the predictive analysis Task in SAMOA core. Because of this preserving streaming analysis architecture for data streams as it is, model is retrained at every instance.

Streaming Clustering with SAMOA

SAMOA Stream Clustering class is also same as the Streaming KMeans Clustering used in Spark based implementation. But the internal functionality is totally different in SAMOA. Not as a initial stage we are directly training stream data with **native stream processing** (or tuple-at-a-time processing) which can be later used for micro-batch processing as well with additional algorithms or ML framework.
Constructor:
StreamingKMeansClustering(int learnType, int windowShift, int numAttributes, int batchSize, double ci, int numClusters, int numIterations, double alpha)

Functions:

Function	Input	Output	Description
cluster()	double[]eventData	Object[]output	Clustering with eventData as double array and if model is updated the send cluster centers back as output Object array.

Streaming Clustering With Samoa WSO2 CEP Siddhi Extension

As same as the previous case, we can use wso2 siddhi extension to feed data into the ML topology as streams.

streamingml:streamclusteringsamoa([learn-type], [window-shift], [batch-size], [number-iterations], [number-clusters], [alpha], [confidence-interval], PE, ATV, V, AP, RH)

Learn-Type

0 - Mini-Batch Processing

1- Mini-Batch Processing with Moving Window with Moving Shift Variable

2 - Time - Window Based Mini-Batch Processing (For Future Implementation)

Window-shift

Only apply when the learn type is 1 - mini-batch processing with the moving window

Number of Clusters

Number of KMeans Clusters need where the output streams contain that number of cluster centers for predictions and analysis.

[3] Where $c_{(t)}$ is the previous center for the cluster, $n_{(t)}$ is the number of points assigned to the cluster thus far, x_t is the new cluster center from the current batch, and $m_{(t)}$ is the number of points added to the cluster in the current batch. The decay factor α can be used to ignore the past: with α=1 all data will be used from the beginning; with α=0 only the most recent data will be used.

Ci (confidence-interval)
Confidence Interval is not using in algorithms for now. Keep it for future as optional field for algorithms.

Query Example

```
@Import('ccppInputStream:1.0.0')
define stream ccppInputStream (PE double, ATV double, V double, AP double, RH double);

@Export('ccppOutputStream:1.0.0')
define stream ccppOutputStream (PE double, ATV double, V double, AP double, RH double,
stderr double, center0 string, center1 string);

from ccppInputStream#streaming:streamclusteringsamoa(0, 0, 1000, 10, 2, 1, 0.95, PE, ATV, V,
AP, RH)
select *
insert into ccppOutputStream;
```

Hint: This query is used for the CCPP dataset from UCI repository.

Input CEP Event Streams

Input stream to the streaming KMeans Clustering extension is consist of the all features of the sample. Since this is unsupervised learning there are no dependent variables.

```
@Import('ccppInputStream:1.0.0')
define stream ccppInputStream (PE double, ATV double, V double, AP double, RH double);
```

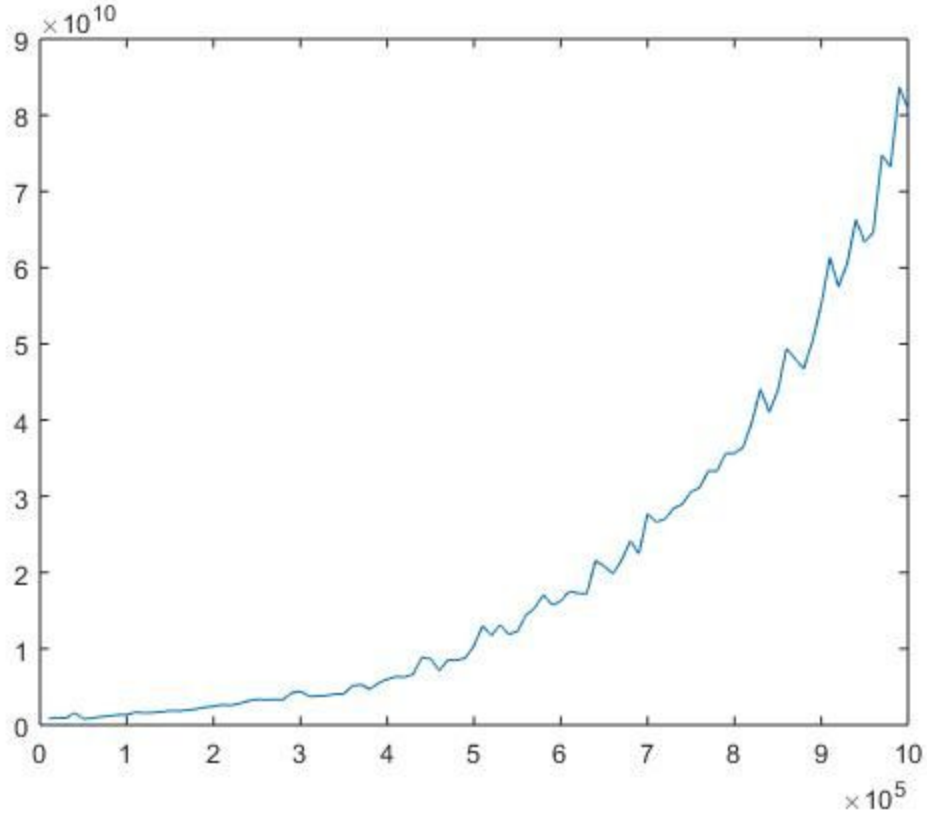
Output CEP Event Streams

Output when there is a new model or updated model will be sent with a WSSSE and cluster centers. As an example if we have k clusters, k+1 the output stream is looks like follow,

[var0, var1, ….., varp, stderr, cluster0, cluster11…., clusterk];

Results and Comparison

Before we planned to got to SAMOA we did couple of performance evaluation of algorithms such as SGD (Stochastic Gradient Descent) . There we can see how the mini-batch processing time vary with the batch-size. Processing time is in nanoseconds and batch size is in number of data events.
Y axis - Mini-Batch Processing time (nanoseconds)
X axis - Mini-Batch Size (number of samples)



Therefore you can see that even when we used 1 million samples point to train it will take only 90 seconds (one and half minutes). Therefore we can see feasibility of applying spark based one for both mini-batch processing as well as streaming leaning. And also after finalizing the implementation and moving towards SAMOA based implementation for better streaming learning we verified the output from two algorithms where we could not find much difference between them. They are almost same as correct results. This is very important because these two implementations use two different algorithms.

Spark based streaming clustering learner
center0:
440.873301276124,
25.477847235065635,
64.20812280377294,
1010.9613963380791,
67.80943221749587

center1:
470.25742216416273,
12.785760940561731,
42.586181145220955,
1015.9444850860008,
79.6946897452645

SAMOA based streaming clustering learner
Center:0:
442.06527377405365,
25.51594930115884,
60.35312500444376,
1010.6246222277139,
61.262047085828065

Center:1:
469.66365199261037,
12.6577054812157,
43.59210147812603,
1014.3543441369945,
76.81973235658958

Here you can see that when we use two clusters with same data set and same batch size are almost same. In clustering implementations results are given as cluster centers.

Challenges & Future Improvements

Integrating SAMOA into the WSO2 current CEP and ML is the challenging part which was never done by someone before. To to that i had to go through the both architectures deeply. Finally i came up with the solution that preserve the streaming learning and processing architecture of the both SAMOA and CEP. One important thing and probably an advantage for the WSO2 is that both SAMOA and CEP distribution can be done with Apache Storm.

After going through couple of architectural reviews i have understand the way that SAMOA works and its architecture. Then i wrote couple of programs based on the SAMOA basic building blocks to make sure i realized their architecture well. Then write couple of programs to make SAMOA work with external environment and with custom external streams. Then build complete streaming clustering stack as new SAMOA topology and then connect it

with our CEP siddhi extension developed for the Streaming Clustering. This is the first project of integrating SAMOA into WSO2 product stack. Therefore i resolved lots of issues that arrived during this period.

As a future improvements, this can be extend to all machine learning algorithms. SAMOA has flexibility of using any ML framework with their architecture. Therefore in the future based on this initial project anyone can easily develop and deployed any custom streaming application with learning and training data.

GSOC Final Evaluation

1. GSOC My Work here https://github.com/dananjayamahesh/GSOC2016).you can find samoa extensions and spark siddhi extension inside gsoc/ folder insode GSOC2016 repo. This contains all the major commits i have done during the GSOC period.
2. streamingml github repository here (https://github.com/dananjayamahesh/streamingml)
3. streamingml integration with carbon-ml master branch here (https://github.com/dananjayamahesh/carbon-ml/tree/master/components/extensions/org.wso2.carbon.ml.siddhi.extension/src/main/java/org/wso2/carbon/ml/siddhi/extension)
4. carbon-ml PR here (https://github.com/wso2/carbon-ml/pull/232)
5. Documentation here

References

[1] http://schradersworld.com/Work/Links_and_Definitions/Big%20Data_files/Big%20Data.htm
[2] http://www.business2community.com/brandviews/upwork/streaming-data-big-data-high-velocity-01621466?utm_content=buffer2aa72&utm_medium=social&utm_source=facebook.com&utm_campaign=buffer#aLcVT0BfClJZxS3w.97

[3] https://spark.apache.org/docs/1.6.2/streaming-programming-guide.html
[4] http://jmlr.csail.mit.edu/papers/volume16/morales15a/morales15a.pdf
[5] https://samoa.incubator.apache.org/
[6] https://samoa.incubator.apache.org/documentation/SAMOA-Topology.html